

Efficient and Flexible Lineage Construction for Probabilistic Databases

VON DER FAKULTÄT FÜR MINT - MATHEMATIK, INFORMATIK, PHYSIK,
ELEKTRO- UND INFORMATIONSTECHNIK DER
BRANDENBURGISCHEN TECHNISCHEN UNIVERSITÄT
COTTBUS – SENFTENBERG

ZUR ERLANGUNG DES AKADEMISCHEN GRADES EINES
DOKTOR DER INGENIEURWISSENSCHAFTEN
(DR.-ING.)

GENEHMIGTE DISSERTATION

VORGELEGT VON
DIPLOM-INFORMATIKER

SEBASTIAN LEHRACK

GEBOREN AM 01.09.1978 IN COTTBUS

Gutachter: Prof. Dr.-Ing. habil. Ingo Schmitt

Gutachter: Prof. Dr. Martin Theobald

Gutachter: Prof. Dr.-Ing. Norbert Ritter

Tag der mündlichen Prüfung: 04.11.2016

Abstract

In contrast to traditional data applications, many real-world scenarios nowadays depend on managing and querying huge volumes of *uncertain* and *incomplete* data. This new type of applications emerge, for example, when we integrate data from various sources, analyse social/biological/chemical networks or conduct privacy-preserving data mining.

A very promising concept addressing this new kind of *probabilistic data applications* has been proposed in the form of *probabilistic databases*. Here, a tuple only belongs to its table or query answer with a specific likelihood. That probability expresses the uncertainty about the given data or the confidence in the answer. The most challenging tasks for probabilistic databases is query evaluation. In fact, there are even simple relational queries for which determining the occurrence probability of a single answer tuple is hard for $\#\mathcal{P}$.

Lineage formulas constitute the central concept under investigation in this work. In short, the mechanism behind lineage formulas facilitates the representation and evaluation of events of the probability space, which is defined by a probabilistic database. On the basis of lineage formulas, we devise a framework that is designed as a combination of a relational database layer and an additional probabilistic query engine.

In particular, the following three aspects are studied:

- an efficient construction of lineage formulas,
- an orthogonal combination of lineage optimization techniques, which are performed within the relational database layer and the probabilistic query engine, and
- effective and compact data structures to represent lineage formulas within a probabilistic query engine.

The developed framework provides a novel lineage construction method that is able to construct nested lineage formulas, to avoid large tuple sets within the relational database layer tuples, and to provide full relational algebra support. In addition, the proposed system completely resolves the conflict between the contradicting query plans optimized for the relational database layer and the probabilistic query engine.

Zusammenfassung

Probabilistische Datenbanken standen in den letzten Jahren im Fokus intensiver Forschungsaktivitäten. Dies wurde durch eine Vielzahl von Anwendungsszenarien motiviert, in denen eine effiziente Verwaltung von großen, unsicheren Datenbeständen unabdingbar ist. Typische Anwendungsgebiete lassen sich leicht in den Bereichen der Datenextraktion und -integration, der wissenschaftlichen Datenauswertung und der Analyse von sensorischen und sozialen Netzwerken finden.

In einer probabilistischen Datenbank wird jedes Datentupel mit einer Eintrittswahrscheinlichkeit annotiert. Diese verdeutlicht, mit welcher Wahrscheinlichkeit das jeweilige Tupel zu einer bestimmten Datentabelle bzw. zu einem berechneten Anfrageergebnis gehört. Für probabilistische Datenbanken ist die Berechnung der Eintrittswahrscheinlichkeit für ein Ergebnistupel die größte Herausforderung, da dieses Problem in der Komplexitätsklasse $\#\mathcal{P}$ liegt. Diese Klasse beinhaltet alle Probleme, die mindestens so schwer sind wie das Zählen aller erfüllenden Modelle einer aussagenlogischen Formel. Es gilt $\mathcal{NP} \subseteq \#\mathcal{P}$.

Traditionell werden Auswertungsverfahren für probabilistische Datenbanken in *intensionale* und *extensionale* Ansätze unterteilt. Intensionale Ansätze greifen auf die Konstruktion und Auswertung von *Abstammungsformeln* zurück. Auf der Basis von Abstammungsformeln, die das zentrale Untersuchungsobjekt dieser Arbeit darstellen, können Ereignisse des Wahrscheinlichkeitsraumes einer probabilistischen Datenbank repräsentiert werden.

In der vorliegenden Dissertation wird ein System entworfen, welches als Kombination einer relationalen Datenbank-Schicht und einer zusätzlichen probabilistischen Auswertungskomponente konzipiert ist. Hierbei werden folgende drei Hauptaspekte untersucht:

- die effiziente Konstruktion von Abstammungsformeln,
- die orthogonale Kombination von Optimierungstechniken und
- die Entwicklung von effektiven und kompakten Datenstrukturen für die Kodierung von Abstammungsformeln für die probabilistische Auswertungskomponente.

Die entwickelten Techniken ermöglichen die schnelle Generierung von geschachtelten Abstammungsformeln, die Vermeidung von großen Tupel-Mengen innerhalb der relationalen Datenbank-Schicht und die Unterstützung aller relationalen Anfrage-Operatoren.

Contents

I	Introduction	9
1	Probabilistic data(base) applications	13
2	Main research questions	15
2.1	Efficient lineage construction	15
2.2	Orthogonal combination of optimizations	16
2.3	Compact data structures for lineage formulas	16
2.4	Probabilistic query engine: Prophecy	17
2.5	Summary	17
3	Outline	19
3.1	Thesis parts	19
3.2	Related works	20
3.3	Experiments	20
3.4	Basic notations and operations	20
II	Fundamentals of Probabilistic Databases	21
4	Probabilistic databases	23
4.1	Basic definition of a probabilistic database	23
4.2	Special classes of probabilistic databases	25
4.3	Summary	27
5	Query semantics	29
5.1	Relational algebra queries	29
5.2	Query semantics of possible answers	30
5.3	Query classes	31
5.4	Summary	32
6	Lineage formulas	33
6.1	Query evaluation with lineage formulas	33
6.2	Classical lineage construction	35
6.3	Syntactic normal forms and standard transformations	38
6.4	Summary	42
7	Related works	43
7.1	Historical overview	43
7.2	Important probabilistic database systems	44
7.3	Extensional evaluation techniques	45
7.4	Intensional evaluation techniques	46
7.5	Further approaches	47
7.6	Summary	48

III	Motivation	49
8	State-of-the-art lineage construction techniques	53
8.1	Why do we use lineage formulas?	53
8.2	Classical construction of nested lineage formulas	55
8.3	Generation of lineage formulas in DNF	55
8.4	Creation of networks for representing lineage formulas	56
8.5	Design goals for lineage construction	59
8.6	Summary	62
9	Vertical lineage construction in a nutshell	63
9.1	Adjusted basic architecture	63
9.2	Overview of theoretical framework	64
9.3	Vertical lineage construction algorithm	69
9.4	Summary	74
IV	Theoretical framework	75
10	Main transformation chain	79
10.1	Main step (1): Processing an equivalent domain calculus query	80
10.2	Main step (2): Setting up relevant conditions over relevant domains	88
10.3	Main step (3): Mapping relevant conditions to lineage formulas	93
10.4	Summary	95
11	Relevant domains	97
11.1	Properties of relevant domains	97
11.2	Generation of relevant domains	105
11.3	Summary	108
12	Advanced aspects of relevant domains	109
12.1	Construction of a relevant domain	109
12.2	Simplified construction rules	114
V	Query processing	119
13	Event relations	123
13.1	Generating event relation \mathbf{E}_Q	123
13.2	First part of \mathbf{E}_Q : all possible answers $Q_{\text{poss}(\mathcal{W})}$	125
13.3	Second part of \mathbf{E}_Q : all relevant domains \mathbf{D}^t	125
13.4	Third part of \mathbf{E}_Q : all required atomic tuple events	126
13.5	Experiments: relational processing	130
13.6	Summary	135
14	Advanced aspects of event relations	137
14.1	Containment of all possible answers $Q_{\text{poss}(\mathcal{W})}$ in \mathbf{E}_Q	137
14.2	Simplifying relation predicates of the form $R(\bar{c}, _)$	138
14.3	Decomposed event relations	142
15	Vertical lineage construction	147
15.1	Vertical lineage construction algorithm	147
15.2	Experiments: lineage construction	155
15.3	Summary	160

16 Advanced aspects of vertical lineage construction	163
16.1 Correctness of vertical lineage construction	163
16.2 Vertical lineage construction algorithm for decomposed event relations	170
17 Lineage optimization	173
17.1 Motivation and existing lineage optimization approaches	173
17.2 Orthogonal combination of lineage optimizations by decoupling	177
17.3 Experiments: lineage optimization and probability computation	180
17.4 Summary	184
18 Advanced aspects of lineage optimization	187
18.1 Minimal relevant domains	188
18.2 Correctness proof for decoupled optimization	199
18.3 Exclusive worlds for uncovered domain classes	201
19 Lineage Factorization and Compression	209
19.1 Existing lineage factorization and compression methods	209
19.2 Lineage factorization by λ -labeling	212
19.3 Optimal λ -labeling	218
19.4 Extended vertical lineage construction	220
19.5 Experiments: compressed data structures	222
19.6 Summary	226
20 Advanced aspects of lineage factorization and compression	227
20.1 Correctness of λ -labeling approach	227
20.2 Optimal subtree labeling	229
VI Conclusions	239
21 Research questions	241
21.1 Lineage construction	241
21.2 Orthogonal combination of lineage optimizations	242
21.3 Compact data structures for lineage formulas	243
A Experimental databases	245
A.1 Data sets	245
A.2 Tested Queries	248
B Basic notations	253
B.1 Restriction of tuples	253
B.2 Overlapping concatenation of tuples	254
B.3 Disjoint union of sets	254

Part I

Introduction

During the last decades relational databases [20, 41] have been established as one of the dominant technologies for storing, querying, and analyzing data in digital information management. Very often, they serve as the central data management layer of a complex information system.

Originally, classical relational databases have been developed to process deterministic data. In such case, each tuple belongs to its table or query answer with absolute certainty. When single attribute values are missing in a tuple, the so-called *null values* can be used to fill these gaps [21, 22].

In contrast to traditional data applications, many real-world scenarios nowadays depend on managing and querying huge volumes of *uncertain* and *incomplete* data. This new type of applications emerge, for example, when we integrate data from various sources, analyse social/biological/chemical networks or conduct privacy-preserving data mining.

A very promising concept addressing this new kind of *probabilistic data applications* has been proposed in the form of *probabilistic databases*. Here, a tuple only belongs to its table or query answer with a specific likelihood. That probability expresses the uncertainty about the given data or the confidence in the answer.

The most challenging task for probabilistic databases is query evaluation. In fact, there are even simple relational queries for which determining the occurrence probability of a single answer tuple is hard for $\#P$ [24, 111]. Problems of this complexity class ask for the number of accepting paths of a non-deterministic Turing machine which runs in polynomial time, i.e., $\#P$ also includes \mathcal{NP} problems.

In the discourse of this thesis we present different techniques which aim to significantly simplify the general query evaluation problem. As a result, we make probabilistic databases more applicable for numerous real world applications.

This study is organized as follows:

- Chapter (1): a set of pragmatic problems and applications that can be tackled with probabilistic data(base) approaches,
- Chapter (2): the main research question we are interested in and
- Chapter (3): the outline of this thesis.

Chapter 1

Probabilistic data(base) applications

In order to highlight the practical benefits of probabilistic databases, we present a selection of application areas where probabilistic data(base) techniques have been successfully applied. The following list of topics were first collected by Panse in [85]:

- **Sensor networks:** In a sensor network, some nodes can unintentionally measure incorrect or noisy data of the objects they observe, e.g., moving objects in spatiotemporal databases [19, 118]. This impreciseness can be captured conveniently by probabilistic data and query models [19, 18, 118, 33].
- **RFID management:** The scanning of RFID tags is a further scenario where noisy data [42, 58, 112] can be properly expressed by adding confidence scores to data and answer tuples.
- **Automatic schema matching:** Algorithms that match different relational schemes are an important research and application field in data management. Despite a lot of effort made in automatic schema matching approaches [90, 99, 10], there is still a need for developing better techniques [40]. An important class of approaches incorporates uncertainty as an essential part of their generated relational and XML schema mappings [76, 31, 119].
- **Duplicate detection:** The elimination of duplicates is another interesting data integration task. Very often, it has to be decided when different data tuples refer to the same real-world entity.

Over the last years, several promising methods have been proposed for detecting duplicates [12, 52, 86] and modeling different possible values within a target database [5, 49, 15, 14].

- **Information extraction from unstructured text:** Systems for information extraction usually need to process huge amounts of unstructured texts. The world wide web in particular provides an almost unlimited collection of such texts contained by millions of web pages. Among others, probabilistic extraction techniques have been successfully employed to deal with the inherently given uncertainty [64, 47, 23, 113].

A prominent example of a probabilistic extraction system called the Never-Ending Language Learner (NELL) [77] was started in January 2010. It is still harvesting new knowledge by identifying structured facts of the form (entity, relation, value) from millions of unstructured web pages. The semantic interpretation of them is usually weighted by score values expressing the assigned degree of confidence.

- **Named entity recognition (NER):** NER tools map different descriptions of specific real-world entities to a controlled vocabulary which represents them [34, 48].
- **Sentiment detection:** Lately, sentiment detection applications have been addressed by probabilistic data and query models as well [111]. Such techniques intend to expose the underlying intention of a given document, text, sentence, or feature. For instance, the expressed opinion can be classified as positive, negative, or neutral.
- **Optimal character recognition (OCR):** OCR algorithms perform electronic conversions of images of typewritten or printed text into machine-encoded text. Typically, they process

printed paper as passport documents, invoices, bank statements, receipts, etc. in order to create an electronic version of these texts/documents.

Apparently, an automatic identification of all words and numbers in their correct forms is a hard task, since handwriting of most people has a very individual character. Probabilistic approaches which can support OCR systems effectively have been proposed by [17, 62].

- **Image object recognition:** In addition to OCR techniques, further stochastic and statistical algorithms focus on identifying natural objects within given electronic images [75, 80].
- **Scientific databases:** Scientific applications often rely on uncertain data, since the measurements of their input data usually suffer from an inevitable technical impreciseness. Probabilistic databases can capture this uncertainty and impreciseness in a very natural way. They have been sustainably utilized in various scientific fields, e.g.:
 - in astronomy [110],
 - protein chemistry data processing [79],
 - prediction of protein functions [29, 121],
 - modeling of protein-protein interactions [88], and
 - verification of uncertainty in biological image analyses [2].
- **Fraud detection:** A reliable warning method for credit card fraud is clearly a very interesting topic for banks. In [108], a technique taking advantage of hidden Markov models has been presented for such applications.
- **Risk assessment and business intelligence:** Several methods for risk assessment and business intelligence also make use of stochastic and statistical methods and models [8, 54, 53, 117, 106].
- **Prediction of future events:** The prediction of future events also needs to deal with a significant degree of uncertainty inherently [89], e.g.:
 - forecasting weather,
 - forecasting traffic on streets,
 - forecasting data networks,
 - predicting prices on stock markets, and
 - predicting future demands of electricity.

These tasks could already benefit from probabilistic data query and data models [116, 95].

Chapter 2

Main research questions

In this section, we pose the main research questions of this thesis and summarize our own contributions. Please note that a more coherent motivation and description of all studied problems and their underlying challenges are laid out in Part (III). Before we can offer more detailed explanations, we need to introduce a series of fundamental concepts and formalisms.

The most important technique laid out in Part (II) is the idea of *lineage formulas*. They constitute the central concept under investigation in this work. In short, the mechanism behind lineage formulas facilitates the representation and evaluation of events of the probability space defined by a probabilistic database.

The main practical outcome of this thesis is a framework that mainly exploits lineage formulas. Our developed framework belongs to a certain class of probabilistic database systems. All members of this class are designed as a combination of a relational database layer and an additional on-top probabilistic query engine.

With this 2-tier basic architecture as a starting point, our work addresses the following three more concrete aspects:

- Section (2.1): an efficient construction of lineage formulas,
- Section (2.2): an orthogonal combination of optimization techniques, which are performed within the relational database layer and the probabilistic query engine, and
- Section (2.3): effective and compact data structures to represent lineage formulas within a probabilistic query engine.

Next, we formulate the corresponding research questions.

2.1 Efficient lineage construction

Our first question directly targets a fast and flexible construction of lineage formulas given that a full relational algebra support is provided.

Question (1): *How can we construct lineage formulas efficiently for all relational algebra queries by means of a 2-tier probabilistic database system?*

Prior to this work, existing approaches basically followed three strategies in order to build lineage formulas:

- creating nested lineage formulas within a single relation attribute by restricting their lengths in advance¹ [37],

¹Here, we assume that the maximal number of tuples of a specific relational table is conceptionally not bounded. In contrast, the length of a given tuple is considered to be limited by the cumulative size of its attribute type sizes.

- building a lineage formula in DNF² within a RDBMS by forbidding relational algebra queries with difference operations [7], and
- generating nested lineage formulas, which are encoded by a large network of referencing tuples within the relational database layer [114, 101].

We devise in this work a novel method that is able to construct nested lineage formulas, to avoid large tuple sets within the relational database layer tuples, and to provide full relational algebra support. It outperforms existing approaches by one order of magnitude during the evaluation of fairly complex queries.

2.2 Orthogonal combination of optimizations

Our second main question concerns the interplay of query plans which are optimized for two competing goals. On one hand, the first type of query plans aims for a fast query evaluation within the relational database layer in a classical way. On the other hand, the technique of safe query plans proposed by [24] directly creates *tractable* lineage formulas, which are easy to evaluate.

Question (2): *How can we orthogonally combine optimization techniques performed within the relational database layer and the probabilistic query engine?*

Previous studies already showed that query plans, which support either the relational processing part or the evaluation of lineage formulas can be very contradicting [114, 82]. Consequently, we had to decide between an RDBMS-oriented query plan, a lineage-optimized query plan, or a hybrid query plan in order to optimize the overall query evaluation [82].

In this thesis, we propose a novel mechanism that completely resolves the conflict between the two contradicting basic types of query plans. On the basis of the relational domain calculus, we can independently generate underlying domains and formula structures in an optimized way without any interferences.

Again, our method outperforms state-of-the-art approaches by one order of magnitude, if we evaluate complex queries.

2.3 Compact data structures for lineage formulas

Last but not least, we explore data structures, which are used to represent lineage formulas within our probabilistic query engine. In order to set up very compact data structures, we are interested in finding lineage *subformulas* that occur multiple times within a set of lineage formulas. Such shared formula structures can be easily exploited to compress the data structures, which encode lineage formulas.

Question (3): *For which query class can we identify all shared lineage subformula efficiently?*

Two prominent studies addressing lineage factorization and compression are presented in [101] and [84]. The first technique essentially relies on factor networks. The corresponding bisimulation algorithm of [101] is capable of pinning down *all* existing shared factors with a quadratic complexity considering the sizes of the given network [101].

Alternatively, the approach of [84] just works on the input query with a constant complexity in the database size. But in contrast to [101], it can only process a small subset of all relational algebra queries effectively.

²The abbreviation DNF stands for disjunctive normal form.

In this study, however, we develop a new factorization and compression mechanism that is applicable on arbitrary relational algebra queries. When we incorporate our lineage factorization and compression ideas into our lineage construction algorithm, for a very large subset of all relational algebra queries we can even guarantee to find all shared lineage subformulas in linear time.

2.4 Probabilistic query engine: Prophecy

All algorithms proposed in this work are implemented in our probabilistic query engine *Prophecy*. Prophecy is meant to be the probabilistic query engine for the next generation of our probabilistic database *ProQua*. An online demonstration of the current version of ProQua can be found at:

<http://dbis.informatik.tu-cottbus.de/ProQua/>

2.5 Summary

In this chapter, we presented our three main research questions under investigation. Moreover, we gave a brief overview of our own contributions. Finally, we introduced our open source library Prophecy, which is the practical outcome of this work.

Chapter 3

Outline

This thesis consists of six main parts:

- Part (I): a summary of our main research questions,
- Part (II): an introduction to all essential definitions and concepts used in this work,
- Part (III): the motivation behind our contributions,
- Part (IV): an outline of our theoretical framework,
- Part (V): the development of our practical techniques, and
- Part (VI): a summary of this work providing our conclusions.

In principle, the first six parts of our work can be read sequentially. They cover all relevant topics and contributions in a self-contained manner. Moreover, Part (V) contains an additional chapter with advanced topics for each core chapter. These additional chapters are not necessary to understand our basic concepts and ideas. They rather provide further theoretical and technical details for more interested readers.

3.1 Thesis parts

In the following, we shortly sketch the main topics of this study offered by its different parts.

Part (II): Fundamentals of probabilistic databases

This section of our thesis presents all definitions and basic mechanisms required to develop and explain our theoretical and practical contributions. It comprises the following chapters:

- Chapter (4): a definition of probabilistic databases built on possible-worlds-semantics,
- Chapter (5): some query semantics used in our framework,
- Chapter (6): lineage formulas, which constitute our central concept of interest, and
- Chapter (7): existing approaches and systems proposed over the last years.

Part (III): Motivation

In Part (III), we explain the unsolved problems of all related approaches that have motivated us to look out for new methods for lineage construction. This part contains:

- Chapter (8): a description and assessment of the challenges and issues that have been already tackled by existing state-of-the-art techniques and
- Chapter (9): a summary of our key concept developed for vertical lineage construction.

Part (IV): Theoretical framework

All algorithms we have developed rely on a solid theoretical foundation, which is devised in Part (IV). It contains the following chapters:

- Chapter (10): conceptional transformation chain describing an alternative way of lineage construction and
- Chapter (11): basic ideas for an efficient and simple generation of relevant domains.

Part (V): Query processing

After laying out all necessary theoretical foundations, we will be prepared to design and devise our more pragmatic techniques. Part (V) includes the following chapters:

- Chapter (13): efficient and simple rules for computing event relations,
- Chapter (15): novel vertical construction algorithm for lineage formulas,
- Chapter (17): concept for an orthogonal combination of optimized query plans,
- Chapter (19): new lineage factorization technique used to build compact data structures.

Part (VI): Conclusions

In the final Part (VI), we summarize this study by the conclusion, which are inferred from our concepts and algorithms developed in this work.

3.2 Related works

In Part (II), we outline the most important works published in the different research fields of probabilistic databases. First and foremost, this description serves as an overview. A more detailed discussion of related approaches can be found in Part (III), in which we study the strengths and weaknesses of existing methods in comparison to our ideas.

In addition, our main chapters of Part (IV) and (V) also contain remarks and references to related approaches, which elucidate and underline similarities and differences.

3.3 Experiments

In order to illustrate the pragmatic benefits of our conceptional ideas, we conducted several series of experiments employing our probabilistic query engine Prophecy. All our experiments were based on three probabilistic databases and two sets of algebra queries. They are described in more detail in Appendix (A). The results of our experiments are directly discussed in the respective chapters of Part (V).

3.4 Basic notations and operations

In Appendix (B), we clarify some basic notations and operations that might be used differently in the literature.

Part II

Fundamentals of Probabilistic Databases

Chapter 4

Probabilistic databases

The following chapter consists of:

- Section (4.1): our basic definition of a probabilistic database based on possible-worlds-semantics and
- Section (4.2): two popular subclasses of probabilistic databases that have been used to set up the investigated data sets in our experiments, see Appendix (A).

4.1 Basic definition of a probabilistic database

In a probabilistic database, several relational database instances are managed and queried simultaneously. Thereby, all instances, also known as *possible worlds*, are defined over the same relational database schema. Among all possible worlds, we can find the one which actually represents our reality. Since that “real world” is assumed to be unknown, we cope with this uncertainty by defining a discrete probability space over all possible worlds. This probability space then embodies our probabilistic database. In particular, we adapt the definition of Suciu, Olteanu, Re, and Koch for our purposes [111].

Definition 4.1 (Probabilistic database). *Let $\mathcal{R} = \{R_1, \dots, R_n\}$ be a relational database schema with n relation names and n corresponding relation schemes.*

- *Then, an incomplete database is defined as a finite set of relational database instances*

$$\mathcal{W} = \{W^1, W^2, \dots, W^k\}.$$

- *A single relational database instance $W \in \mathcal{W}$, also called a possible world, consists of a set of finite relation instances for R_1, \dots, R_n . Since we denote a relation instance of R that is given in a specific world W as $R(W)$, we can express an incomplete database as*

$$\mathcal{W} = \left\{ \underbrace{\{R_1(W^1), \dots, R_n(W^1)\}}_{W^1}, \dots, \underbrace{\{R_1(W^k), \dots, R_n(W^k)\}}_{W^k} \right\}.$$

- *A probabilistic database is a discrete probability space $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ over an incomplete database \mathcal{W} such that $\mathbf{P} : \mathcal{W} \rightarrow [0, 1]$ is an additive probability measure with*

$$\sum_{W \in \mathcal{W}} \mathbf{P}(W) = 1.$$

- *We additionally define that all worlds have a strict positive probability, i.e., $\forall W \in \mathcal{W} : \mathbf{P}(W) > 0$.*

$R_1(W^{max})$		$R_2(W^{max})$			$R_3(W^{max})$			$R_4(W^{max})$	
	A		A	B		A	B		A
t_1	1	t_3	1	3	t_5	1	3	t_9	1
t_2	2	t_4	1	4	t_6	1	4	t_{10}	2
					t_7	2	5	t_{11}	3
					t_8	2	6		

$R_1(\hat{W})$		$R_2(\hat{W})$			$R_3(\hat{W})$			$R_4(\hat{W})$	
	A		A	B		A	B		A
t_1	1	t_4	1	4	t_5	1	3	t_{10}	2
t_2	2				t_6	1	4		
					t_8	2	6		

$R_1(W^\emptyset)$		$R_2(W^\emptyset)$			$R_3(W^\emptyset)$			$R_4(W^\emptyset)$	
	A		A	B		A	B		A
-	-	-	-	-	-	-	-	-	-

Figure 4.1: Possible worlds W^{max} , \hat{W} and W^\emptyset of Example (4.1) on Page (24)

In order to exemplify our ideas and concepts, we take advantage of the following running example of a probabilistic database.

Example 4.1. [Running example database] In this study, we always set up a probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ by deriving the set of all possible worlds \mathcal{W} from the set of all possible tuples. This set includes all tuples that are principally possible within a single specific world. For our running example, the set of all possible tuples is given by $\{t_1, \dots, t_{11}\}$. Those tuples are specified and contributed, as shown in the relation instances $R_1(W^{max}), \dots, R_4(W^{max})$ of Figure (4.1) on Page (24).

According to possible-worlds-semantics, we do not know which tuples out of $\{t_1, \dots, t_{11}\}$ actually occur in reality. Thus, we define a probabilistic database capturing all $2^{11} = 2048$ possible tuple combinations. By that, we obtain a set of 2048 possible relational database instances. They form our set of all possible worlds:

$$\mathcal{W} = \{W^\emptyset, W^1, \dots, \hat{W}, \dots, W^{2046}, W^{max}\}.$$

Every world $W \in \mathcal{W}$ contains one specific relation instance for R_1, \dots, R_4 , i.e.,

$$\mathcal{W} = \left\{ \underbrace{\{R_1(W^\emptyset), \dots, R_4(W^\emptyset)\}}_{W^\emptyset}, \dots, \underbrace{\{R_1(\hat{W}), \dots, R_4(\hat{W})\}}_{\hat{W}}, \dots, \underbrace{\{R_1(W^{max}), \dots, R_4(W^{max})\}}_{W^{max}} \right\}.$$

Figure (4.1) on Page (24) shows three possible worlds taken from \mathcal{W} . We already referred to the relation instances of world W^{max} . They represent the world with the maximal set of tuples. The opposite world of W^{max} is also shown in Figure (4.1) on Page (24). The world W^\emptyset only includes empty relation instances, i.e.,

$$R_1(W^\emptyset) = R_2(W^\emptyset) = R_3(W^\emptyset) = R_4(W^\emptyset) = \emptyset.$$

The third example of a world we present here is the non-extreme world \hat{W} in Figure (4.1) on Page (24). In \hat{W} , the tuples from $\{t_1, t_2, t_4, t_5, t_6, t_8, t_{10}\}$ are present and the tuples of $\{t_3, t_7, t_9, t_{11}\}$ are absent.

Besides the set of all possible worlds \mathcal{W} , the Definition (4.1) on Page (23) also involves the probability measure \mathbf{P} over \mathcal{W} in order to specify a probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$. Since our techniques are completely independent from a specific probability measure \mathbf{P} , we do not need to fix \mathbf{P} for our running example. We purposely leave out a more detailed specification of \mathbf{P} at this point.

4.2 Special classes of probabilistic databases

Next, we introduce two prominent probabilistic database classes that constituted the base for our experiments:

- probabilistic *tuple-independent* databases (*TID databases*) and
- probabilistic *block-independent-disjoint* databases (*BID databases*).

Both classes are defined by the interrelations of their involved *atomic tuple events*.

Definition 4.2 (Atomic tuple event). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database built over the relational database schema $\mathcal{R} = \{R_1, \dots, R_n\}$.*

- *Then, each tuple t over a relation schema is uniquely associated with an atomic tuple event e_t defined over the probability space $(\mathcal{W}, \mathbf{P})$, i.e., $e_t \subseteq \mathcal{W}$.*
- *An atomic tuple events e_t collects all worlds where the tuple t occurs:*

$$e_t := \{W \in \mathcal{W} \mid t \in R(W)\}.$$

- *The marginal probability over all worlds*

$$\mathbf{P}(e_t) := \sum_{W \in \mathcal{W}: t \in W} \mathbf{P}(W)$$

describes the likelihood that the tuple t actually exists in reality.

- *We call the marginal probability $\mathbf{P}(e_t)$ as occurrence probability of tuple t .*

Tuple-independent databases (TID databases)

In a tuple-independent database, all atomic tuple events are assumed to be independent from each other, i.e.,

$$\forall t, \hat{t} : (t \neq \hat{t}) \Rightarrow \mathbf{P}(e_t \wedge e_{\hat{t}}) = \mathbf{P}(e_t) * \mathbf{P}(e_{\hat{t}}).$$

This means that each tuple t can exist in a specific world W independent from the presence or absence of another tuple \hat{t} :

$$\forall t, \hat{t} : (t \neq \hat{t}) \Rightarrow (\mathbf{P}(t \in W) = \mathbf{P}(t \in W \mid \hat{t} \in W) = \mathbf{P}(t \in W \mid \hat{t} \notin W)).$$

In this case, the set of all possible worlds \mathcal{W} is always given as the power set of the basic set of tuples given in W^{max} , i.e., $\mathcal{W} = 2^{W^{max}}$.

In a TID database, we can compute the probability of a specific world $W \in \mathcal{W}$ by simply multiplying the probabilities of all atomic tuple events according to the presence/absence of their associated tuples in the world W :

$$\forall W \in \mathcal{W} : \mathbf{P}(W) = \prod_{t \in W} \mathbf{P}(e_t) * \prod_{t \notin W} (1 - \mathbf{P}(e_t)).$$

Definition 4.3 (TID database). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database. We say \mathbf{pdb} is a tuple-independent database (TID database), if*

$$\forall t, \hat{t} \in \bigcup_{R \in \mathcal{R}} R(W^{max}) : (t \neq \hat{t}) \Rightarrow (\mathbf{P}(e_t \wedge e_{\hat{t}}) = \mathbf{P}(e_t) * \mathbf{P}(e_{\hat{t}}))$$

holds.

Definition (4.3) on Page (25) implies that a TID database can be easily defined by its set of all possible tuples, i.e., all tuples in W^{max} , when each possible tuple $t \in W^{max}$ is additionally annotated by its occurrence probability $\mathbf{P}(e_t)$. We give concrete examples of TID databases in Part (III) and Appendix (A).

Block-independent-disjoint databases (BID databases)

Besides TID databases, we used a further special type of probabilistic databases known as *block-independent-disjoint databases* (BID databases). They are popular due to their capability to express attribute uncertainty very intuitively.

The main idea of BID databases is to partition their sets of all possible tuples into different disjoint blocks. Thereby, a single block B cannot span more than one relation. Moreover, it is required that all tuples *within* a block B are associated with *disjoint* atomic tuple events:

$$\forall t, \hat{t} \in B : (t \neq \hat{t}) \Rightarrow \mathbf{P}(e_t \wedge e_{\hat{t}}) = 0.$$

Please be aware that *one* tuple of each block can maximally occur in a specific world. That is guaranteed by the disjointness property of their atomic tuple events. On the other hand, atomic tuple events from *different* blocks are still *independent* from each other. Similarly to TID databases, atomic tuple events from different blocks do not affect the presence/absence of their tuples.

To set up the characteristic blocks of a BID database, we make use of the so-called *event keys* which are specified in the form of attribute sets. They are part of every BID relation schema. Taking event keys into account, we define that a block is formed by all tuples, which share the *same values* for the attributes of the given event keys.

Definition 4.4 (BID database). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database.*

- *If we introduce an event key for each relation schema R , i.e., $ekey(R) \subseteq attr(R)$, we can define partitions over the set of all possible tuples given by W^{max} :*

$$\forall R \in \mathcal{R} : R(W^{max}) = B_{R,1} \dot{\cup} \dots \dot{\cup} B_{R,m}$$

with

$$\forall t, \hat{t} : (t, \hat{t} \in B_R) \Leftrightarrow ((t, \hat{t} \in R(W^{max})) \wedge (t_{ekey(R)} = \hat{t}_{ekey(R)})).$$

- *Then, we call \mathbf{pdb} a block-independent-disjoint database (BID database), if*

$$\forall t \in B, \hat{t} \in \hat{B} : ((B = \hat{B}) \wedge (t \neq \hat{t})) \Rightarrow \mathbf{P}(e_t \wedge e_{\hat{t}}) = 0$$

and

$$\forall t \in B, \hat{t} \in \hat{B} : ((B \neq \hat{B}) \wedge (t \neq \hat{t})) \Rightarrow \mathbf{P}(e_t \wedge e_{\hat{t}}) = \mathbf{P}(e_t) * \mathbf{P}(e_{\hat{t}})$$

hold.

An example of a BID database is given in Appendix (A) in the form of one of our experimental databases. Finally, it is necessary to point out that a TID database is automatically a BID database, if all event keys $ekey(R)$ equal their corresponding attribute sets, i.e., $\forall R \in \mathcal{R} : ekey(R) := attr(R)$. In that case, each possible tuple forms its own block:

$$\forall R \in \mathcal{R} : R(W^{max}) = \{t_1, \dots, t_m\} = \underbrace{\{t_1\}}_{B_{R,1}} \dot{\cup} \dots \dot{\cup} \underbrace{\{t_m\}}_{B_{R,m}}.$$

4.3 Summary

In this chapter, we provided a basic definition of a probabilistic database. All our ideas and techniques rely on this definition, which is based on the well-known concept of possible-worlds-semantics. In addition, two special cases of probabilistic database namely tuple-independent databases (TID databases) and block-independent-disjoint databases (BID databases) have been shortly introduced.

Chapter 5

Query semantics

After outlining our data model in the previous chapter, we now turn our attention to the query model we intend to use. This chapter consists of the following sections:

- Section (5.1): the query language of our framework,
- Section (5.2): the query semantics our framework follows, and
- Section (5.3): a series of further query (sub)classes.

5.1 Relational algebra queries

Relational algebra [20] has without a doubt been one of the most important database query languages in the last decades. All commercial vendors of relational database management systems facilitate relational algebra queries as the core of their own pragmatic SQL query languages.

Despite encountering various query languages in this study, all our considered input queries can be expressed as an equivalent algebra query.

Definition 5.1 (Algebra query). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database built over the relational database schema $\mathcal{R} = \{R_1, \dots, R_n\}$.*

- *Then, an algebra query Q is recursively defined by following standard algebra operators [20, 74], if Q_1 and Q_2 are also representing relational algebra (sub)queries:*
 - *relation operator: $Q = R$, where $R \in \mathcal{R}$,*
 - *selection operator: $Q = \sigma_F(Q_1)$, where F is a propositional formula,*
 - *projection operator: $Q = \pi_{\mathcal{A}}(Q_1)$ with its set of projection attributes $\mathcal{A} \subseteq \text{head}(Q_1)$,*
 - *join operator: $Q = Q_1 \bowtie Q_2$ obeying the semantics of a natural join,*
 - *union operator: $Q = Q_1 \cup Q_2$ requiring compatible relation schemes,*
 - *difference operator: $Q = Q_1 \setminus Q_2$ requiring compatible relation schemes and*
 - *renaming operator: $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$, where the attributes of \mathcal{A} are replaced by the attributes of \mathcal{B} .*
- *Additionally, we define the auxiliary function $\text{head}(Q)$. It gives the set of all output attributes of Q .*
- *We explicitly allow that the attribute set \mathcal{A} of a projection operation $\pi_{\mathcal{A}}(Q_1)$ can be empty. For such an operation, the empty tuple $t = ()$ is returned, if and only if Q_1 is not empty:*

$$\pi_{\emptyset}(Q_1) := \begin{cases} \{()\} & \text{if } Q_1 \neq \emptyset \\ \emptyset & \text{else.} \end{cases}$$

We introduce here three example queries, which are based on our running example database shown in Example (4.1) on Page (24).

Example 5.1 (Running example queries). *In order to illustrate our main ideas, we declare the three example queries Q_{\bowtie} , Q_{\cup} , and Q_{\setminus} that are applied on our running example database shown in Example (4.1) on Page (24):*

$$\begin{aligned} Q_{\bowtie} &:= \pi_A((R_1 \bowtie R_2) \bowtie (R_1 \bowtie R_3)) \\ Q_{\cup} &:= \pi_A(\sigma_{(B=4)}(R_2)) \cup \pi_A(R_1 \bowtie R_2) \\ Q_{\setminus} &:= \pi_A(R_1 \bowtie R_3) \setminus \pi_A(\sigma_{(A=1)}(R_3)). \end{aligned}$$

The query labels Q_{\bowtie} , Q_{\cup} and Q_{\setminus} highlight the main operation within the respective queries. They apparently cover all important algebra operators, if we take into account that selection and projection operations are partly involved in all of them.

5.2 Query semantics of possible answers

In order to process an algebra query on a given probabilistic database, we employ the query semantics known as *possible answers* [111].

Definition 5.2 (Query semantics). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database and Q be an algebra query.*

- *When we apply a query Q on the relation instances of a specific world $W \in \mathcal{W}$, we write $Q(W)$ and say that Q is evaluated in world W .*
- *A tuple t is called a possible answer to Q , if there exists at least one world $W \in \mathcal{W}$ such that $t \in Q(W)$.*
- *The set of all possible answers $Q_{\text{poss}(\mathcal{W})}$ is then defined as*

$$Q_{\text{poss}(\mathcal{W})} := \{t \mid \exists W \in \mathcal{W} : t \in Q(W)\} = \bigcup_{W \in \mathcal{W}} Q(W).$$

- *In addition, we determine the marginal probability over all worlds for a specific possible answer tuple t :*

$$\mathbf{P}(t \in Q) := \sum_{W \in \mathcal{W} : t \in Q(W)} \mathbf{P}(W).$$

- *We call the marginal probability $\mathbf{P}(t \in Q)$ of a tuple t as its answer probability.*
- *The query evaluation result of Q on \mathbf{pdb} then consists of the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$ and a function $pr_Q : Q_{\text{poss}(\mathcal{W})} \rightarrow [0, 1]$ returning all corresponding answer probabilities:*

$$Q(\mathbf{pdb}) = (Q_{\text{poss}(\mathcal{W})}, pr_Q) \quad \text{with} \quad \forall t \in Q_{\text{poss}(\mathcal{W})} : pr_Q(t) := \mathbf{P}(t \in Q).$$

Definition (5.2) on Page (30) implies that we can determine the query evaluation result $Q(\mathbf{pdb})$ by means of the following three steps:

- Step (1): running the query Q on each world of $\mathcal{W} = \{W^1, \dots, W^k\}$ separately using standard relational algebra semantics, i.e.,

$$Q(W^1), \dots, Q(W^k),$$

- Step (2): building $Q_{\text{poss}(\mathcal{W})}$ as a union of all answer tuples determined by these independent runs:

$$Q_{\text{poss}(\mathcal{W})} := \bigcup_{W \in \mathcal{W}} Q(W) = Q(W^1) \cup \dots \cup Q(W^k),$$

and

- Step (3): calculating the answer probabilities $\mathbf{P}(t \in Q)$ of all answer tuples using the results of the computed runs and the given world probabilities:

$$\forall t \in Q_{\text{poss}(\mathcal{W})} : pr_Q(t) := \mathbf{P}(t \in Q) = \sum_{W \in \mathcal{W} : t \in Q(W)} \mathbf{P}(W).$$

Example 5.2 (Conceputal query evaluation on probabilistic database). *If we evaluate our example query Q_{\bowtie} in the world $\hat{W} = \{t_1, t_2, t_4, t_5, t_8, t_{10}\}$ of Example (4.1) on Page (24), we obtain $Q_{\bowtie}(\hat{W}) = \{(1)_{(A)}\}$ as a result. Contrarily, the same query applied on W^0 delivers an empty result, i.e., $Q_{\bowtie}(W^0) = \emptyset$.*

After evaluating Q_{\bowtie} on all 2048 possible worlds, we unify all computed answer sets and achieve $Q_{\bowtie, \text{poss}(\mathcal{W})} = \{(1)_{(A)}\}$ as the final set of all possible answers.

Lastly, we calculate the answer probability of our only answer tuple $(1)_{(A)}$:

$$pr_Q((1)_{(A)}) = \mathbf{P}((1)_{(A)} \in Q_{\bowtie}) = \sum_{W \in \mathcal{W} : (1)_{(A)} \in Q_{\bowtie}(W)} \mathbf{P}(W)$$

by summing up the probabilities of all worlds W , where $(1)_{(A)}$ appears in a determined query result $Q_{\bowtie}(W)$ with $W \in \mathcal{W}$. Among others, the probabilities $\mathbf{P}(W^{\max})$ and $\mathbf{P}(\hat{W})$ contribute to $\mathbf{P}((1)_{(A)} \in Q_{\bowtie})$, since $(1)_{(A)} \in Q_{\bowtie}(W^{\max})$ and $(1)_{(A)} \in Q_{\bowtie}(\hat{W})$.

Please be aware that Definition (5.2) on Page (30) first and foremost provides us with the semantics for our query evaluation. From a more pragmatic point of view, we know that we cannot run a query in all worlds, because the number of all possible worlds grows exponentially, i.e., $|\mathcal{W}| = 2^{|W^{\max}|}$, if it embraces all possible tuple combinations derived from the tuples in W^{\max} .

Fuhr and Röllecke tackled this problem in their milestone paper [39] through a novel technique called lineage formulas. The efficient construction of these formulas is the central subject of their study. Before we explain lineage formulas in more detail in Chapter (6), we introduce different query (sub)classes also used in the next chapters.

5.3 Query classes

Many approaches that have been developed for querying probabilistic databases are tailored only for a limited set of algebra queries, see Chapter (7). Very often, the usage of a certain relational operator is completely forbidden. In contrast, our techniques do not possess such restrictions. They all provide *full* algebra support. This means that they can work on arbitrary queries formed by all relational standard operators: selection, projection, join, union, difference, and renaming.

Similarly to [57], we denote the query class allowing all algebra operators by the acronym **SPJUD**. Starting from **SPJUD**, we define a set of further query *subclasses* using different criteria.

Definition 5.3 (Query subclasses of **SPJUD**). *Let **SPJUD** be the set of all relational algebra queries. Then, we build subsets of **SPJUD** by taking the following four criteria into account:*

- **Set of involved operators:** *Most importantly, we consider query subclasses that are specifically defined over a certain subset of operators. The set of permitted operators corresponds to the abbreviated operator names of its class label.*

- **Repeating relations:** Moreover, we differentiate between the so-called repeating and non-repeating queries. A query Q is said to be non-repeating, if a specific relation is not used more than once in Q . Otherwise, Q is said to be repeating. We label a class of non-repeating queries with the prefix **nr**. Contrarily, query class labels for repeating queries have no prefix.
- **Data complexity of query evaluation:** Finally, we can classify queries based on their data complexity [111]. If the computation of all answer probabilities $\mathbf{P}(t \in Q)$ is in $\#\mathcal{P}$ -complete, then we say the considered query Q is a hard query. On the contrary, a query Q possessing answer probabilities that can be evaluated in polynomial time is called tractable.

Example 5.3 (Query subclasses). Respecting Definition (5.3) on Page (31), we can give following example query classes:

- **Set of involved operators:** Queries from **SPJ** are just constructed from selection, projection, join, and renaming operations.
- **Repeating relations:** The query label **nrSPJ** describes the class of non-repeating queries just built from selection, projection, join, and renaming operations. Our example queries Q_{\bowtie} , Q_{\cup} , and Q_{\setminus} of Example (5.1) on Page (30) are taken from **SPJUD**. They all are repeating queries.
- **Data complexity of query evaluation:** In [26], all tractable **SPJU**-queries on **TID** databases are precisely defined by the proposed dichotomy.

5.4 Summary

In the last chapter, we first declared relational algebra as our basic query language. Afterwards, the query semantics used in our approach was defined. Lastly, we gave several criteria that can be used to address various query (sub)classes derived from the general class **SPJUD**.

Chapter 6

Lineage formulas

In this chapter, we describe the basics of our main concept under investigation, namely *lineage formulas*. It contains the following sections:

- Section (6.1): how lineage formulas can simplify query evaluation on probabilistic databases,
- Section (6.2): classical rules used to construct lineage formulas, and
- Section (6.3): standard normal forms and standard transformations applicable on lineage formulas.

6.1 Query evaluation with lineage formulas

The query semantics of Definition (5.2) on Page (30) involves the function of all answer probabilities

$$pr_Q : Q_{\text{poss}(\mathcal{W})} \rightarrow [0, 1] \quad \text{with} \quad \forall t : pr_Q(t) := \mathbf{P}(t \in Q)$$

as the second part of our query result $Q(\mathbf{pdb})$. We already pointed out that Definition (5.2) on Page (30) cannot be practically implemented, since the set of all possible worlds \mathcal{W} can grow exponentially in input size. Instead, we can calculate the probabilities of $pr_Q(t)$ by means of lineage formulas.

Definition 6.1 (Lineage formula). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database.*

- *Then, a lineage formula φ is a finite propositional formula constructed from the set of atomic tuple events $\{e_{t_1}, \dots, e_{t_n}, T, F\}$ of \mathbf{pdb} and the logical standard operators \wedge, \vee , and \neg .*
- *In a lineage formula, we consider the truth values T and F as atomic tuple events. They embody the sure event*

$$T := \mathcal{W} \quad \text{with} \quad \mathbf{P}(T) = \mathbf{P}(\mathcal{W}) = 1$$

and the impossible event

$$F := \emptyset \quad \text{with} \quad \mathbf{P}(F) = \mathbf{P}(\emptyset) = 0.$$

- *In addition to the standard operators \wedge, \vee , and \neg , we introduce three further logical operators denoted as \oslash, \oslash , and \oplus :*
 - *The operator symbols \oslash and \oslash stand for a n -ary conjunction or disjunction operation. Their semantics are defined by a respective sequences of $(n - 1)$ nested binary conjunction or disjunction operations:*

$$(e_1 \oslash \dots \oslash e_n) :\Leftrightarrow (e_1 \wedge (e_2 \wedge (\dots \wedge e_n)))$$

$$(e_1 \oslash \dots \oslash e_n) :\Leftrightarrow (e_1 \vee (e_2 \vee (\dots \vee e_n))).$$

- We use \oslash/\odot in prenex and infix form:

$$(e_1 \oslash \dots \oslash e_n) \Leftrightarrow \left(\bigodot_{\hat{\varphi} \in \{e_1, \dots, e_n\}} \hat{\varphi} \right)$$

$$(e_1 \odot \dots \odot e_n) \Leftrightarrow \left(\bigodot_{\hat{\varphi} \in \{e_1, \dots, e_n\}} \hat{\varphi} \right).$$

- Lastly, we also employ a particular n -ary exclusive-or operator \oplus . It expresses a disjunctive combination of mutually exclusive subformulas:

$$(e_1 \oplus \dots \oplus e_n) :\Leftrightarrow (\forall i \in \{1, \dots, n\} : (e_i \Rightarrow (\forall j \in (\{1, \dots, n\} \setminus \{i\}) : e_j \equiv F))).$$

Example 6.1 (Lineage formula). Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database, where e_1, \dots, e_n are atomic tuple events of the probability space $(\mathcal{W}, \mathbf{P})$. Then, the following propositional formulas represent lineage formulas of \mathbf{pdb} :

$$\begin{aligned} \varphi_1 &= (e_1 \wedge e_2) \vee \neg e_3 \\ \varphi_2 &= (e_1 \oslash e_2 \oslash e_3) \vee \neg(e_1 \odot e_2 \odot e_3) \\ \varphi_3 &= \left(\bigodot_{\hat{\varphi} \in \{e_1, e_2, e_3\}} \hat{\varphi} \right) \vee \neg \left(\bigodot_{\hat{\varphi} \in \{e_1, e_2, e_3\}} \hat{\varphi} \right) \\ \varphi_4 &= (e_1 \oplus e_2 \oplus e_3) \wedge \neg e_3 \\ \varphi_5 &= \left(\bigoplus_{\hat{\varphi} \in \{e_1, e_2, e_3\}} \hat{\varphi} \right) \wedge \neg e_3. \end{aligned}$$

In Definition (6.1) on Page (33), we base our lineage formulas on atomic tuple events. In principle, each atomic tuple event of Definition (4.2) on Page (25) defines an event of the probability space $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$. This suggests that we can always consider tuple events as subsets of \mathcal{W} . This perspective directly corresponds to the standard definition of an event of a given probability space. On the other hand, atomic tuple events are also often described indirectly by means of binary random variables. Next, we briefly introduce both variants.

Definition 6.2 (Modeling of tuple events within lineage formulas). Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database and let φ^t be a lineage formula given for a specific tuple t .

- **Atomic tuple events e_t directly given as subsets of \mathcal{W} :** From probability theory, we know that each subset of \mathcal{W} defines an event of $(\mathcal{W}, \mathbf{P})$. So, a specific atomic tuple event e_t describes the (sub)set of worlds, where its associated tuple t is present:

$$e_t := \{W \in \mathcal{W} \mid \exists R \in \mathcal{R} : t \in R(W)\}.$$

Accordingly, a lineage formula stands for the event that is built when we directly interpret all atomic tuple events e_t in φ^t as world sets and map the logical operators to their respective set operations ($\wedge \leftrightarrow \cap$, $\vee \leftrightarrow \cup$ and $\neg \leftrightarrow \setminus$).

- **Atomic tuple events e_t indirectly modeled by binary random variables over \mathcal{W} :** Intensional evaluation algorithms often work on lineage formulas built from a set of binary random variables, e.g., [37, 83]. In that case, an atomic tuple event e_t is represented by a random variable

$$X_t : \mathcal{W} \rightarrow \{T, F\},$$

which is defined as

$$X_t(W) := \begin{cases} T & \text{if } \exists R \in \mathcal{R} : t \in R(W) \\ F & \text{else.} \end{cases}$$

It can be easily used to determine a corresponding event from $(\mathcal{W}, \mathbf{P})$:

$$e_t := \{W \in \mathcal{W} \mid X_t(W) = T\}.$$

For the sake of brevity, we often write

$$X_t(W) \quad \text{instead of} \quad (X_t(W) = T).$$

If we combine random variables by logical operators, then we obviously achieve complex events of the probability space $(\mathcal{W}, \mathbf{P})$.

Both methods explained in Definition (6.2) on Page (34) are able to represent events from $(\mathcal{W}, \mathbf{P})$, we can choose between them according to our current needs. We already mentioned that algorithms often abstract the underlying probability space $(\mathcal{W}, \mathbf{P})$ by addressing tuple events in the form of random variables. They are easier to handle than sets of worlds.

We take advantage of both forms in the following sections. In our more theoretical discussions, we investigate events directly, which are given as subsets of \mathcal{W} . On the contrary, the inputs and outputs of our algorithms are usually processed as binary random variables. Normally, we simply write e_t as a short form for an atomic tuple event without giving a concrete representation type.

By referring to our second interpretation of Definition (6.2) on Page (34), we can define the probability of a lineage formula $\mathbf{P}(\varphi^t)$ as follows:

Definition 6.3 (Probabilities and equivalence of lineage formulas). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database and φ and $\hat{\varphi}$ be two lineage formulas, where atomic tuple events are indirectly modeled as binary random variables.*

- *Then, the probability of φ is determined to*

$$\mathbf{P}(\varphi) = \sum_{W \in \{W \in \mathcal{W} \mid \varphi(W)\}} \mathbf{P}(W).$$

- *A lineage formula φ is said to be tractable, if and only if its probability can be determined in polynomial time considering the maximal possible set of tuples given in W^{max} .*
- *The two lineage formulas φ and $\hat{\varphi}$ are equivalent, if and only if $\varphi(W) \Leftrightarrow \hat{\varphi}(W)$ in all worlds $W \in \mathcal{W}$:*

$$\varphi \equiv \hat{\varphi} \quad :\Leftrightarrow \quad (\forall W \in \mathcal{W} : \varphi(W) \Leftrightarrow \hat{\varphi}(W)).$$

In practice, we compute the probability $\mathbf{P}(\varphi)$ by standard evaluation algorithms, e.g., [37, 83, 28].

6.2 Classical lineage construction

Fuhr and Röllecke showed in [39] that the answer probability $\mathbf{P}(t \in Q)$ of a tuple t equals the probability of a lineage formula, i.e.:

$$\mathbf{P}(t \in Q) = \mathbf{P}(\varphi^t),$$

if the lineage formula φ^t is built for the tuple t with a certain set of rules.

Lemma 6.1 (Classical construction rules for a lineage formula φ_Q^t). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database and Q be an algebra query. Then, a lineage formula φ_Q^t is recursively*

$q_1 = R_1$			$q_2 = R_3$			$q_3 = R_1 \bowtie R_3$			$q_4 = \sigma_{A=1}(R_3)$		
A	$\varphi_{q_1}^t$		A	B	$\varphi_{q_2}^t$	A	B	$\varphi_{q_3}^t$	A	B	$\varphi_{q_4}^t$
1	e_1		1	3	e_5	1	3	$(e_1 \wedge e_5)$	1	3	e_5
2	e_2		1	4	e_6	1	4	$(e_1 \wedge e_6)$	1	4	e_6
			2	5	e_7	2	5	$(e_2 \wedge e_7)$			
			2	6	e_8	2	6	$(e_2 \wedge e_8)$			

$q_5 = \pi_A(R_1 \bowtie R_3)$			$q_6 = \pi_A(\sigma_{A=1}(R_3))$		
A	$\varphi_{q_5}^t$		A	$\varphi_{q_6}^t$	
1	$((e_1 \wedge e_5) \oslash (e_1 \wedge e_6))$		1	$(e_5 \oslash e_6)$	
2	$((e_2 \wedge e_7) \oslash (e_2 \wedge e_8))$				

$Q_{\setminus} = \pi_A(R_1 \bowtie R_3) \setminus \pi_A(\sigma_{A=1}(R_3))$		
A	$\varphi_{Q_{\setminus}}^t$	
1	$((e_1 \wedge e_5) \oslash (e_1 \wedge e_6)) \wedge \neg(e_5 \oslash e_6)$	
2	$((e_2 \wedge e_7) \oslash (e_2 \wedge e_8)) \wedge \neg(\text{F})$	

Figure 6.1: Lineage construction process for the example query Q_{\setminus}

constructed by following rules:

$$\begin{aligned}
Q = R & : \varphi_Q^t := \begin{cases} e_t & \text{if } t \in R(W^{max}) \\ F & \text{else} \end{cases} \\
Q = \sigma_F(Q_1) & : \varphi_Q^t := \varphi_{Q_1}^t \\
Q = \pi_A(Q_1) & : \varphi_Q^t := \bigoplus_{(i \in Q_1(W^{max}), i_A=t)} \varphi_{Q_1}^t \\
Q = Q_1 \bowtie Q_2 & : \varphi_Q^t := \varphi_{Q_1}^{t_1} \wedge \varphi_{Q_2}^{t_2} \\
Q = Q_1 \cup Q_2 & : \varphi_Q^t := \varphi_{Q_1}^t \vee \varphi_{Q_2}^t \\
Q = Q_1 \setminus Q_2 & : \varphi_Q^t := \varphi_{Q_1}^t \wedge \neg(\varphi_{Q_2}^t) \\
Q = \rho_{(A \leftarrow B)}(Q_1) & : \varphi_Q^t := \varphi_{Q_1}^t.
\end{aligned}$$

After constructing all lineage formulas, we can determine the answer probability for each answer tuple t as

$$\mathbf{P}(t \in Q) = \mathbf{P}(\varphi^t).$$

Proof. See [39]. □

The rules of the last lemma define how we can set up a single lineage formula based on a given algebra query Q . To generate all relevant lineage formulas, we can easily exploit those rules as well. Therefore, we extend all possible tuples given in $R_1(W^{max}), \dots, R_n(W^{max})$ with their respective atomic tuple events and apply the rules of Lemma (6.1) on Page (35) on these augmented relations.

Example 6.2 (Lineage formulas of running example queries). *The classical construction rules of Lemma (6.1) on Page (35) produce the following lineage formulas for our queries Q_{\bowtie} , Q_{\cup} , and Q_{\setminus} from Example (5.1) on Page (30):*

$$\begin{aligned}
\varphi_{\bowtie}^{(1)} &= ((e_1 \wedge e_3) \wedge (e_1 \wedge e_5)) \oslash ((e_1 \wedge e_4) \wedge (e_1 \wedge e_6)) \\
\varphi_{\cup}^{(1)} &= e_4 \vee ((e_1 \wedge e_3) \oslash (e_1 \wedge e_4))
\end{aligned}$$

$$\begin{aligned}\varphi_{\setminus}^{(1)} &= ((e_1 \wedge e_5) \oslash (e_1 \wedge e_6)) \wedge \neg(e_5 \oslash e_6) \\ \varphi_{\setminus}^{(2)} &= ((e_2 \wedge e_7) \oslash (e_2 \wedge e_8)) \wedge \neg(F).\end{aligned}$$

In Figure (6.1) on Page (36), we can follow the stepwise construction for the last two lineage formulas $\varphi_{\setminus}^{(1)}$ and $\varphi_{\setminus}^{(2)}$ in all details. Thereby, the intermediate construction steps are based on the subqueries

$$\begin{aligned}q_1 &:= R_1 & q_3 &:= R_1 \bowtie R_3 & q_5 &:= \pi_A(R_1 \bowtie R_3) \\ q_2 &:= R_3 & q_4 &:= \sigma_{A=1}(R_3) & q_6 &:= \pi_A(\sigma_{A=1}(R_3))\end{aligned}$$

of

$$Q_{\setminus} = \pi_A(R_1 \bowtie R_3) \setminus \pi_A(\sigma_{A=1}(R_3)).$$

Next, we determine the maximal length of a lineage formula, if it is constructed as suggested in Lemma (6.1) on Page (35). We define the length of φ_Q^t as the total number of atomic tuple events¹ in φ_Q^t . It is denoted by $|\varphi_Q^t|$.

Lemma 6.2 (Maximal length of a constructed lineage formula). *Let Q be a query of **SPJUD** with the length $|Q|$ and φ_Q^t be a lineage formula generated by the classical construction rules of Lemma (6.1) on Page (35). The query Q is applied on a probabilistic database **pdb** that consists of all possible tuple combinations of $R_1(W^{max}), \dots, R_m(W^{max})$. Then, the length of φ_Q^t is bounded by*

$$|\varphi_Q^t| \leq |Q| * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}.$$

Proof. First of all, we assume that Q contains no projection operations, i.e., $Q \in \mathbf{SJUD}$. In that subcase, the rules of Lemma (6.1) on Page (35) imply that each relation R of Q contributes one atomic tuple event at most to a final lineage formula, since all operators concatenate a maximum of two subformulas. Therefore, we achieve a maximal length of $|Q|$ for each generated lineage formula.

Second of all, we know from relational algebra that the result set of a query from **SJUD** cannot have more than

$$\max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$$

tuples [74].

By taking also projection operations into account, we can combine all lineage subformulas into the single lineage formula $\varphi^{()}$. This longest lineage formula $\varphi^{()}$ is constructed by a final projection $\pi_{\emptyset}(Q)$. Because we have at most

$$\max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$$

lineage formulas of length $|Q|$ in Q , we finally obtain

$$|Q| * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$$

as the maximal length for the longest lineage formula $\varphi^{()}$. □

Please note that a lineage formula can also be built for a tuple which is not a possible answer in the sense of Definition (5.2) on Page (30).

¹For instance, the length of $\varphi^t = ((e_1 \wedge e_2) \vee e_1)$ is given by $|\varphi^t| = 3$.

Example 6.3 (Lineage formula for non-answer tuple). *Let us have a deeper look at our example query Q_{\setminus} and the tuple $(1)_{(A)}$. It is not a possible answer for Q_{\setminus} , because there is no world W where $(1)_{(A)} \in Q_{\setminus}(W)$. However, the construction rules of Lemma (6.1) on Page (35) build a non-trivial lineage formula*

$$\varphi_{\setminus}^{(1)} = ((e_1 \wedge e_5) \odot (e_1 \wedge e_6)) \wedge \neg(e_5 \odot e_6).$$

Please be aware that $\varphi_{\setminus}^{(1)}$ is not satisfiable, since $\varphi_{\setminus}^{(1)}$ can be simplified to F independently from all other involved atomic tuple events:

$$\begin{aligned} \varphi_{\setminus}^{(1)} &= ((e_1 \wedge e_5) \odot (e_1 \wedge e_6)) \wedge \neg(e_5 \odot e_6) \\ &\equiv ((e_1 \wedge e_5) \odot (e_1 \wedge e_6)) \wedge (\neg(e_5) \wedge \neg(e_6)) \\ &\equiv (e_1 \odot e_5 \odot \neg e_5 \odot \neg e_6) \odot (e_1 \odot e_6 \odot \neg e_5 \odot \neg e_6) \\ &\equiv (e_1 \odot F \odot \neg e_6) \odot (e_1 \odot \neg e_5 \odot F) \\ &\equiv F \odot F \equiv F. \end{aligned}$$

As a result, the lineage and answer probability of tuple $(1)_{(A)}$ equals zero:

$$\mathbf{P}((1)_{(A)} \in Q_{\setminus}) = \mathbf{P}(\varphi_{\setminus}^{(1)}) = \mathbf{P}(F) = \mathbf{P}(\emptyset) = 0.$$

6.3 Syntactic normal forms and standard transformations

In the following, we briefly repeat syntactic standard forms and standard transformations for propositional formulas. They also play an important role in the creation of lineage formulas. We present the following topics in more detail:

- lineage formulas in disjunctive/conjunctive normal form (DNF/CNF),
- lineage formulas given in nested form,
- the one-occurrence form (1OF), and
- the Shannon expansion rule.

Disjunctive/conjunctive normal form (DNF/CNF) for lineage formulas

The disjunctive/conjunctive normal form (DNF/CNF) is a well-known syntactic pattern for propositional and first-order formulas [74].

Definition 6.4 (Disjunctive/conjunctive normal form (DNF/CNF)). *Let φ^t be a lineage formula. Then, φ^t is given in disjunctive/conjunctive normal form (DNF/CNF), if and only if it is built as a disjunction/conjunction of conjunctive/disjunctive clauses. A conjunctive/disjunctive clause is constructed from a finite set of negated/non-negated atomic tuple events, which are all conjunctively/disjunctively combined.*

In general, it is possible to convert each lineage formulas into a disjunctive/conjunctive normal form. However, such a transformation can cause an exponential expansion of the considered lineage formula.

Example 6.4 (Unfolding negated lineage formulas). *The length of the following negated lineage formula triple, if we rewrite it into DNF:*

$$\begin{aligned} \neg((e_1 \wedge e_2 \wedge e_3) \vee (e_4 \wedge e_5 \wedge e_6)) &\equiv ((\neg e_1 \vee \neg e_2 \vee \neg e_3) \wedge (\neg e_4 \vee \neg e_5 \vee \neg e_6)) \\ &\equiv (\neg e_1 \oslash \neg e_4) \oslash (\neg e_1 \oslash \neg e_5) \oslash (\neg e_1 \oslash \neg e_6) \oslash \\ &\quad (\neg e_2 \oslash \neg e_4) \oslash (\neg e_2 \oslash \neg e_5) \oslash (\neg e_2 \oslash \neg e_6) \oslash \\ &\quad (\neg e_3 \oslash \neg e_4) \oslash (\neg e_3 \oslash \neg e_5) \oslash (\neg e_3 \oslash \neg e_6). \end{aligned}$$

Example 6.5 (Lineage formulas in DNF). *Our four lineage formulas from Example (6.2) on Page (36) have the following DNFs:*

$$\begin{aligned} \varphi_{\bowtie}^{(1)} &= ((e_1 \wedge e_3) \wedge (e_1 \wedge e_5)) \oslash ((e_1 \wedge e_4) \wedge (e_1 \wedge e_6)) \\ &\equiv (e_1 \oslash e_3 \oslash e_5) \oslash (e_1 \oslash e_4 \oslash e_6) \\ \varphi_{\cup}^{(1)} &= e_4 \vee ((e_1 \wedge e_3) \oslash (e_1 \wedge e_4)) \\ &\equiv e_4 \oslash (e_1 \oslash e_3) \oslash (e_1 \oslash e_4) \\ \varphi_{\setminus}^{(1)} &= ((e_1 \wedge e_5) \oslash (e_1 \wedge e_6)) \wedge \neg(e_5 \oslash e_6) \\ &\equiv (e_1 \oslash e_5 \oslash \neg e_5 \oslash \neg e_6) \oslash (e_1 \oslash e_6 \oslash \neg e_5 \oslash \neg e_6) \\ \varphi_{\setminus}^{(2)} &= ((e_2 \wedge e_7) \oslash (e_2 \wedge e_8)) \wedge \neg(F) \\ &\equiv (e_2 \oslash e_7 \oslash \neg F) \oslash (e_2 \oslash e_8 \oslash \neg F). \end{aligned}$$

Nested lineage formulas

Besides lineage formulas in DNF/CNF, we are also interested in more irregularly structured lineage formulas called *nested lineage formulas*.

Definition 6.5 (Nested lineage formula). *Let φ^t be a lineage formula. Then, we say that φ^t is nested, if and only if φ^t is not given in DNF/CNF.*

Obviously, the lineage formulas $\varphi_{\bowtie}^{(1)}, \dots, \varphi_{\setminus}^{(2)}$ from Example (6.2) on Page (36) are initially built in a nested form.

One-occurrence form (1OF)

Furthermore, we explore a further normal form called *one-occurrence form* (1OF) [83].

Definition 6.6 (One-occurrence form (1OF)). *Let φ^t be a lineage formula constructed from the atomic tuple events $\{e_1, \dots, e_n, T, F\}$. Then, φ^t is given in one-occurrence form (1OF), if and only if each atomic tuple event out of $\{e_1, \dots, e_n\}$ does not occur more than once in φ^t .*

Example 6.6 (Lineage formulas in 1OF). *According to our last definition, the lineage formulas $\varphi_{\bowtie}^{(1)}, \dots, \varphi_{\setminus}^{(2)}$ in Example (6.2) on Page (36) are initially not generated in 1OF. But we can*

rewrite them into 1OF as follows:

$$\begin{aligned}
\varphi_{\bowtie}^{(1)} &= ((e_1 \wedge e_3) \wedge (e_1 \wedge e_5)) \odot ((e_1 \wedge e_4) \wedge (e_1 \wedge e_6)) \\
&\equiv e_1 \wedge ((e_3 \wedge e_5) \odot (e_4 \wedge e_6)) \\
\varphi_{\cup}^{(1)} &= e_4 \vee ((e_1 \wedge e_3) \odot (e_1 \wedge e_4)) \\
&\equiv e_4 \vee (e_1 \wedge (e_3 \odot e_4)) \\
\varphi_{\setminus}^{(1)} &= ((e_1 \wedge e_5) \odot (e_1 \wedge e_6)) \wedge \neg(e_5 \odot e_6) \\
&\equiv F \\
\varphi_{\setminus}^{(2)} &= ((e_2 \wedge e_7) \odot (e_2 \wedge e_8)) \wedge \neg(F) \\
&\equiv e_2 \wedge (e_7 \odot e_8).
\end{aligned}$$

For TID databases in particular, lineage formulas in 1OF have convenient implications.

Lemma 6.3 (Evaluation of lineage formulas in 1OF). *Let φ^t be a lineage formula evaluated on a TID database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$. If φ^t is in 1OF, the probability $\mathbf{P}(\varphi^t)$ can be computed in linear time considering the length of φ^t . To do so, we take advantage of the standard aggregation rules for probabilities of independent events:*

$$\begin{aligned}
\varphi^t = e_t &: \mathbf{P}(\varphi^t) := \mathbf{P}(e_t) \\
\varphi^t = \varphi_1^t \wedge \varphi_2^t &: \mathbf{P}(\varphi^t) := \mathbf{P}(\varphi_1^t) * \mathbf{P}(\varphi_2^t) \\
\varphi^t = \varphi_1^t \vee \varphi_2^t &: \mathbf{P}(\varphi^t) := \mathbf{P}(\varphi_1^t) + \mathbf{P}(\varphi_2^t) - (\mathbf{P}(\varphi_1^t) * \mathbf{P}(\varphi_2^t)) \\
&= 1 - (1 - \mathbf{P}(\varphi_1^t)) * (1 - \mathbf{P}(\varphi_2^t)) \\
\varphi^t = \neg \varphi_1^t &: \mathbf{P}(\varphi^t) := 1 - \mathbf{P}(\varphi_1^t).
\end{aligned}$$

Proof. See [102, 96]. □

In Part (III), we demonstrate the application of Lemma (6.3) on Page (40) by calculating several lineage probabilities.

Last but not least, we emphasize that a transformation into 1OF is not always possible [111].

Example 6.7 (Lineage formulas without 1OF). *The lineage formula*

$$\varphi^t = (e_1 \wedge e_2) \odot (e_2 \wedge e_3) \odot (e_3 \wedge e_4)$$

is not converted into 1OF by any possible folding operation. Thus, neither

$$\varphi^t \equiv (e_2 \wedge (e_1 \vee e_3)) \odot (e_3 \wedge e_4) \quad \text{nor} \quad \varphi^t \equiv (e_1 \wedge e_2) \odot (e_3 \wedge (e_2 \vee e_4))$$

lead to a 1OF.

Shannon expansion

Another interesting syntactic structure emerges, when we make use of the well-known *Shannon expansion rule* [103]:

$$\varphi^t \equiv (e_i \wedge \varphi_{\langle e_i | \mathbf{T} \rangle}^t) \oplus (\neg e_i \wedge \varphi_{\langle e_i | \mathbf{F} \rangle}^t).$$

The denotation $\varphi_{\langle e_i | T \rangle}^t / \varphi_{\langle e_i | F \rangle}^t$ stands for a syntactic substitution operation², which replaces all appearances of e_i within φ^t by T/F .

If e_i is the only atomic tuple event given more than once in φ^t , the two subformulas $(e_i \wedge \varphi_{\langle e_i | T \rangle}^t)$ and $(\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t)$ are given in 1OF. Then, we can evaluate them as follows.

$$\begin{aligned} \mathbf{P}(\varphi^t) &= \mathbf{P}((e_i \wedge \varphi_{\langle e_i | T \rangle}^t) \oplus (\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t)) \\ &= \mathbf{P}(e_i \wedge \varphi_{\langle e_i | T \rangle}^t) + \mathbf{P}(\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t) - \mathbf{P}((e_i \wedge \varphi_{\langle e_i | T \rangle}^t) \wedge (\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t)) \\ &= \mathbf{P}(e_i \wedge \varphi_{\langle e_i | T \rangle}^t) + \mathbf{P}(\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t) - \mathbf{P}(F) \\ &= \mathbf{P}(e_i \wedge \varphi_{\langle e_i | T \rangle}^t) + \mathbf{P}(\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t) - 0 \\ &= \mathbf{P}(e_i \wedge \varphi_{\langle e_i | T \rangle}^t) + \mathbf{P}(\neg e_i \wedge \varphi_{\langle e_i | F \rangle}^t). \end{aligned}$$

In case of a lineage formula with more than one atomic tuple event that occurs multiple times, we must apply the Shannon expansion rule repeatedly. This implies that the length of the evolving lineage formula doubles in each expansion step. Consequently, the calculation of $\mathbf{P}(\varphi^t)$ using Shannon expansion can demand exponential costs, if we consider the length of the initial lineage formula.

Despite its high complexity, the Shannon expansion rule reached a large popularity, since it provides a technique that can be always employed to evaluate *arbitrary* queries on TID databases [37].

Lemma 6.4. *Let \mathbf{pdb} be a TID database and φ^t be a lineage formula. Then, the lineage probability $\mathbf{P}(\varphi^t)$ can be determined in two steps. First, we create an expanded lineage formula $\hat{\varphi}^t := \text{Shannon}(\varphi^t)$ using*

$$\text{Shannon}(\varphi^t) := \begin{cases} \varphi^t & \text{if } \varphi^t \text{ in 1OF} \\ (e_i \wedge \text{Shannon}(\varphi_{\langle e_i | T \rangle}^t)) \oplus (\neg e_i \wedge \text{Shannon}(\varphi_{\langle e_i | F \rangle}^t)) & \text{else.} \end{cases}$$

Subsequently, we calculate $\mathbf{P}(\hat{\varphi}^t)$ with following rules:

$$\begin{aligned} \hat{\varphi}^t = e_t & : \mathbf{P}(\hat{\varphi}^t) := \mathbf{P}(e_t) \\ \hat{\varphi}^t = \hat{\varphi}_1^t \wedge \hat{\varphi}_2^t & : \mathbf{P}(\hat{\varphi}^t) := \mathbf{P}(\hat{\varphi}_1^t) * \mathbf{P}(\hat{\varphi}_2^t) \\ \hat{\varphi}^t = \hat{\varphi}_1^t \vee \hat{\varphi}_2^t & : \mathbf{P}(\hat{\varphi}^t) := 1 - (1 - \mathbf{P}(\hat{\varphi}_1^t)) * (1 - \mathbf{P}(\hat{\varphi}_2^t)) \\ \hat{\varphi}^t = \hat{\varphi}_1^t \oplus \dots \oplus \hat{\varphi}_n^t & : \mathbf{P}(\hat{\varphi}^t) := \sum_{i \in \{1, \dots, n\}} \mathbf{P}(\hat{\varphi}_i^t) \\ \hat{\varphi}^t = \neg \hat{\varphi}_1^t & : \mathbf{P}(\hat{\varphi}^t) := 1 - \mathbf{P}(\hat{\varphi}_1^t). \end{aligned}$$

Proof. See [37]. □

Example 6.8 (Shannon expansion rules). *By following Lemma (6.4) on Page (41), we first rewrite the lineage formula $\varphi_{\setminus}^{(1)}$ from Example (6.2) on Page (36) into $\hat{\varphi}_{\setminus}^{(1)}$:*

$$\begin{aligned} \varphi_{\setminus}^{(1)} &= ((e_1 \wedge e_5) \otimes (e_1 \wedge e_6)) \wedge \neg(e_5 \otimes e_6) \\ &\equiv \left(e_1 \wedge ((T \wedge e_5) \otimes (T \wedge e_6)) \wedge \neg(e_5 \otimes e_6) \right) \oplus \left(\neg e_1 \wedge ((F \wedge e_5) \otimes (F \wedge e_6)) \wedge \neg(e_5 \otimes e_6) \right) \\ &\equiv e_1 \wedge (e_5 \otimes e_6) \wedge \neg(e_5 \otimes e_6) \end{aligned}$$

²We define the syntactic substitution of subformulas in more detail in Chapter (10).

$$\begin{aligned}
&\equiv \left(e_5 \wedge (e_1 \wedge (T \oplus e_6) \wedge \neg(T \oplus e_6)) \right) \oplus \left(\neg e_5 \wedge (e_1 \wedge (F \oplus e_6) \wedge \neg(F \oplus e_6)) \right) \\
&\equiv \left(e_5 \wedge (e_1 \wedge (e_6 \wedge \neg e_6)) \right) \oplus \left(\neg e_5 \wedge (e_1 \wedge (e_6 \wedge \neg e_6)) \right) \\
&\equiv e_6 \wedge (e_5 \wedge (e_1 \wedge (T \wedge \neg T))) \oplus \neg e_5 \wedge (e_1 \wedge (T \wedge \neg T)) \oplus \\
&\quad \neg e_6 \wedge (e_5 \wedge (e_1 \wedge (F \wedge \neg F))) \oplus \neg e_5 \wedge (e_1 \wedge (F \wedge \neg F)) \\
&= \varphi_{\setminus}^{(1)}.
\end{aligned}$$

Subsequently, the probability $\mathbf{P}(\hat{\varphi}_{\setminus}^{(1)})$ can be determined as follows

$$\begin{aligned}
\mathbf{P}(\hat{\varphi}_{\setminus}^{(1)}) &= \mathbf{P}(e_6) * (\mathbf{P}(e_5) * (\mathbf{P}(e_1) * (\mathbf{P}(T) * (1 - \mathbf{P}(T)))) + \\
&\quad (1 - \mathbf{P}(e_5)) * (\mathbf{P}(e_1) * (\mathbf{P}(T) * (1 - \mathbf{P}(T)))) + \\
&\quad (1 - \mathbf{P}(e_6)) * (\mathbf{P}(e_5) * (\mathbf{P}(e_1) * (\mathbf{P}(F) * (1 - \mathbf{P}(F)))) + \\
&\quad (1 - \mathbf{P}(e_5)) * (\mathbf{P}(e_1) * (\mathbf{P}(F) * (1 - \mathbf{P}(F)))) \\
&= \mathbf{P}(e_6) * (\mathbf{P}(e_5) * (\mathbf{P}(e_1) * (1 * (1 - 1)))) + \\
&\quad (1 - \mathbf{P}(e_5)) * (\mathbf{P}(e_1) * (1 * (1 - 1))) + \\
&\quad (1 - \mathbf{P}(e_6)) * (\mathbf{P}(e_5) * (\mathbf{P}(e_1) * (0 * (1 - 0)))) + \\
&\quad (1 - \mathbf{P}(e_5)) * (\mathbf{P}(e_1) * (0 * (1 - 0))) \\
&= 0 + 0 + 0 + 0 = 0
\end{aligned}$$

6.4 Summary

In this chapter, we described the foundations of lineage formulas. We first discussed query evaluation with lineage formulas. Afterwards, we presented the classical rules for lineage construction proposed by Fuhr and Röllecke. Furthermore, a handful of standard normal forms and standard transformations for propositional and first-order formulas, which are also applicable on lineage formulas, have been revised.

Chapter 7

Related works

Throughout this thesis, we discuss all significant works that are related to our own contributions, e.g., [39, 24, 91, 25, 7, 82, 37, 84, 101]. References to these studies are included in the main chapters of our work. Additionally, we subsequently give a more general overview of all significant systems and approaches. This chapter consists of four sections, which cover the following subjects:

- Section (7.2): a short history of the most important early works,
- Section (7.3): extensional evaluation approaches,
- Section (7.4): intensional evaluation approaches [87], and
- Section (7.5): probabilistic top-k ranking algorithms and probabilistic scoring functions.

We already pointed to an excellent monograph published by Suciu, Olteanu, Re, and Koch [111]. It covers the whole field of evaluation techniques, which rely on the possible answer query semantics as presented in Definition (5.2) on Page (30). Besides [111], Ilyas and Soliman gave in [50] a comprehensive overview of probabilistic ranking techniques.

7.1 Historical overview

This section presents a short overview of study areas that have been developed in the first period of probabilistic database research. We discuss the following contributions in more detail:

- incomplete database systems,
- probabilistic database models and
- probabilistic graphical models.

Incomplete database systems

The necessity of dealing with uncertainty was first recognized by Codd [20, 21]. He studied *null values* already in his initial works on the relational data model. Nowadays, modern relational query languages, e.g., SQL, incorporate null values as a central feature.

Imielinski and Lipski introduced in their groundbreaking work [51] the ideas of conditional tables and strong representation systems. From that point on, the expressiveness and complexity of different uncertainty models for incomplete databases have been intensively investigated, e.g., [1, 46, 45, 73, 81, 59].

Probabilistic database models

Interestingly, the first publications about probabilistic databases were published shortly after the works on the deterministic relational database model were introduced. For example, early publications such as [16, 43, 44, 66] already formulated the idea of modeling uncertain attributes

in form of random variables. Notably, Barbará, Garcia-Molina and Porter demonstrated in [9] a concept of attribute-level uncertainty, which is nowadays known from BID databases.

Beyond that, pioneering works already aimed for a combination of database technologies and information retrieval (IR) techniques. In this area, Fuhr and Rölleke as well as Zimanyi devised probabilistic data models inspired by IR concepts. These models basically augment the possible-worlds-semantics [38, 39, 120].

In [72, 68], we proposed an IR extension of the traditional possible-worlds-semantics as well. More concretely, we introduced a combined retrieval model relying on possible-worlds-semantics and a geometric similarity measure. Our model is the first one to enable the evaluation of complex logic-based similarity conditions on uncertain relational data.

Probabilistic Graphical Models

Besides the database and IR community, researchers from artificial intelligence (AI) have also influenced new developments in the field of probabilistic databases. Probabilistic graphic models in particular (PGM) are strongly related to probabilistic databases [111].

Sen and Deshpande first discussed the relationship between probabilistic databases and PGMs [100]. At the same time, Antova, Koch, and Olteanu presented probabilistic databases basically representing flat Bayesian Networks [6].

More generally, PGMs have been studied in various directions, including AI, (bio)statistics, information theory, and others [4]. Not surprisingly, several excellent monographs addressing PGMs have been published over the last decades, e.g., [87, 27, 61].

Beyond systems strictly obeying the traditional possible-worlds-semantics, alternative types of probabilistic database systems have been also proposed. They often adjust the original possible-worlds-semantics in order to guarantee a tractable query evaluation [65, 30].

7.2 Important probabilistic database systems

In this section, we outline all important probabilistic database systems developed over the last decade, see Figure (7.1). In particular, we intend to highlight their most characteristic features.

- **MystiQ:** Started as one of the first modern projects, the *MystiQ system* extends an RDBMS in order to manage uncertain data in the form of probabilistic databases [94]. It essentially relies on safe plans, which were first laid out in the groundbreaking paper by Dalvi and Suciu [24].

More specifically, MystiQ is able to generate a safe plan for each tractable **nrSPJ**-query applied on a TID/BID database. To evaluate a given safe plan, it pushes its entire probability computation into the relational database layer.

When using MystiQ, lineage formulas are in fact not constructed and evaluated. Instead, MystiQ applies various user-defined functions within the RDBMS in order to calculate the desired answer probabilities.

For a hard query, MystiQ can only approximate the respective answer probabilities.

- **Trio:** In contrast to MystiQ, the *Trio system* [3] extensively exploits the ideas behind lineage formulas for providing probabilistic data management and inference. One of Trio's key features is its query language that offers a special type of predicates. They can be used to formulate conditions directly over the internally constructed lineages.
- **MayBMS/SPROUT/SPROUT2:** The *MayBMS system* [60] is designed as a combination of a customized PostGres DBS and its probabilistic query engine *SPROUT* [82]. It is capable of computing exact and approximated answer probabilities for **SPJU**-queries on TID/BID databases. Please note that the MayBMS system was re-implemented (from [6] to [7, 60]). In this work, we only consider the second version, as described in [7, 60, 82]. For probability computation, SPROUT implements decomposition trees (d-trees) [83] and a special on-the-fly optimization/evaluation algorithm tailored for tractable **nrSPJ**-queries on TID databases [82].

Probabilistic database systems and query engines		
system/engine	key feature	main publications
MystiQ	safe plans	[24]
Trio	lineage predicates	[11, 28]
MayBMS	lineage formulas in DNF	[7]
SPROUT	approximation, on-the-fly lineage optimization	[83, 82]
SPROUT2	full algebra support	[37]
PrDB	probabilistic graphical models	[101]
Orion	continuous probability measures	[105]
ProQua	logic-based scoring functions	[67]

Figure 7.1: Overview of probabilistic database systems and frameworks based on an RDBMS

Dichotomies for algebra query classes on TID/BID databases		
query class	database type	publication
nrSPJ	TID	[24]
nrSPJ\leq	TID	[81]
nrSPJ	BID	[25]
SPJU	TID	[26]
nrSJD	TID	[36]

Figure 7.2: Dichotomies for different query classes

The second version of SPROUT, called *SPROUT2*, utilizes a flexible evaluation algorithm, which enables full algebra support on TID databases [37].

- **PrDB:** The *PrDB system* mainly takes advantage of probabilistic graphical models in order to facilitate complex correlations between tuple events [101]. In particular, it constructs compressed factor networks and applies a specialised probabilistic interference algorithm on them.
- **Orion:** The *Orion database system* accommodates attribute and tuple uncertainty with arbitrary correlations among tuple events [104]. Notably, it is capable of handling discrete *and* continuous probabilistic density functions as special attribute types. Orion is directly integrated into an extended PostGres DBS [109].
- **ProQua:** Our probabilistic database system *ProQua* combines IR concepts with database technologies [70]. In contrast to all other systems, its query language is capable of incorporating complex logic-based similarity conditions and two weighting approaches working on two different operator levels.

7.3 Extensional evaluation techniques

The main principle of an extensional evaluation approach involves rewriting the given input into a so-called *safe plan*. Usually, a safe plan can be directly evaluated within the relational database layer given that the standard SQL operators are extended by a couple of user-defined functions. They are responsible for a correct probability calculation. An additional dedicated probabilistic query engine on top is then not necessary any more.

In addition to easy implementation, extensional approaches have also made extraordinary theoretical contributions. Most importantly, they have established a series of dichotomies which form different query classes with regards to their data complexity.

Figure (7.2) on Page (45) gives an overview of all notable dichotomies published over the last years. The most significant ones can be summarized as follows:

- **Dichotomies for nrSPJ-queries on TID/BID databases:** In [24, 91, 25], Dalvi, Suciu, and Ré presented some of the first important studies. More concretely, they proposed two algorithms which were able to generate query plans for an efficient evaluation of **nrSPJ**-

queries on TID/BID databases. Relying on those algorithms, they proved two dichotomies for identifying all tractable **nrSPJ**-queries on TID/BID databases [24, 25].

- **Dichotomy for SPJU-queries on TID databases:** In [26], Dalvi and Suciu considerably extended their basic algorithms originally presented in [24, 91, 25]. The enhanced versions of their algorithms created safe plans for repeating and non-repeating **SPJU**-queries on TID/BID databases.

By exploiting incidence algebras, they could even establish a dichotomy for **SPJU**-queries on TID databases. A dichotomy for the more complicated case of **SPJU**-queries on BID databases is still open.

- **Dichotomy for nrSPJD-queries on TID databases:** Fink and Olteanu augmented in [36] the original dichotomy for **nrSPJ**-queries from Dalvi and Suciu to non-repeating queries, which involve difference operations on TID databases.

7.4 Intensional evaluation techniques

In contrast to extensional approaches, an *intensional evaluation technique* consists of two processing parts. In the first step, it constructs a data structure that encodes tuple events of the queried probabilistic database. In the second step, the probabilities of the stored tuple events are calculated by applying specialised evaluation algorithms. They usually exploit certain properties of the employed data structures.

We already introduced lineage formulas as a prominent example of an intensional method. Please recall that we can construct lineage formulas for all queries from **SPJUD**, see Lemma (6.1) on Page (35). In conjunction with the Shannon expansion (Section (6.3)), they provide us with a method for evaluating each **SPJUD**-query on an arbitrary TID/BID database.

Lineage formulas in DNF were first used for the MayBMS system [7]. Besides decomposition trees (D-trees), the SPROUT algorithm and the first version of our ProQua system work with lineage formulas given in DNF [83, 82, 70].

The construction of lineage formulas in nested form was demonstrated by Fuhr and Rölleke in their introductory paper [39]. The query engine SPROUT2 makes explicit use of nested lineage formulas [37].

As mentioned earlier, intensional evaluation methods also incorporate the calculation of all encoded answer probabilities. Despite this task generally being a $\#\mathcal{P}$ -hard problem for TID/BID databases [24], several evaluation algorithms have been successfully developed:

- **Decomposition trees:** In [83], Olteanu, Huang, and Koch introduced a deterministic approximation algorithm with error guarantees for computing all answer probabilities. Their algorithm can process **SPJ**-queries on TID/BID databases.

Their algorithm carries out an incremental compilation of lineage formulas into tree structures using Shannon expansion, independence partitioning, and product factorization. The lower and upper probability bounds are thereby calculated and checked against the allowed error during each decomposition step.

- **On-the-fly lineage optimization and evaluation:** SPROUT is the probabilistic query engine developed for the MayBMS system [7, 82]. SPROUT exploits an evaluation algorithm, which rewrites lineage formulas from DNF into 1OF and evaluates them on-the-fly. Only tractable **SPJ**-queries on TID databases can be processed by this technique.

- **Shannon expansion implemented by a masking algorithm:** Another notable evaluation algorithm enhances the probabilistic query engine SPROUT2 [37].

The main feature of SPROUT2 is the ability to evaluate nested lineage formulas without unfolding them into DNF. For this purpose, it makes use of an interesting masking algorithm relying on the Shannon expansion rule. By doing so, it is possible to compute the lower and upper probability bounds for lineage formulas given in a nested form.

- **Transforming lineage formulas into 1OF:** In Section (6.3), we already underlined the important role of lineage formulas given in one-occurrence-form (1OF), if we consider TID/BID databases:

- Sen, Deshpande, and Getoor presented in [102] an algorithm, which ensures the construction of lineage formulas in 1OF, whenever such forms exist.
- Similarly to [102], Roy, Perduca, and Tannen published in [96] an alternative method, which also rewrites a given lineage formula into 1OF, whenever such a transformation is possible.

Both approaches are just processing **nrSPJ**-queries applied on TID databases [102, 96].

- **Ordered binary decision diagrams:** Beyond lineage formulas, there are also other data structures used for encoding tuple events of a probabilistic database. One of the first techniques in this direction was devised by Olteanu and Huang in [81]. They proposed an efficient algorithm for confidence computation based on ordered binary decision diagrams (OBDDs). The sizes of the constructed diagrams were linear in the number of lineage variables. Interestingly, in this work they first introduced the prominent *hierarchical* property for **nrSPJ**-queries on TID databases [111].
- **Knowledge compilation:**
 - In a broader context, Jha and Suciu showed in [56] the compilation of tuple events associated with **SPJU**-queries into 1OF, OBDDs, free binary decision diagrams (FBDDs), and formulas in deterministic decomposable negation normal form (d-DNNF).
 - Additionally, they examined in [55] the problem of calculating the probabilities of Boolean functions represented by OBDDs.
 - Fink, Han, and Olteanu also followed the basic ideas of knowledge compilation in order to propose an evaluation method for **SPJU**-queries with aggregate operations on TID databases [35]. The key idea of their algorithm is a mapping between semimodule/semiring expressions and decomposition trees. The applied probability calculation can be performed in polynomial time, when the size of a compiled tree is considered.

7.5 Further approaches

Apart from classical extensional and intensional evaluation techniques, many other interesting works have been published over the last years:

- **Top-k answers and ranking:** First of all, the computation of the top-k answer set as well as the ranking of all answer tuples drew notable attention.
 - One of the first methods proposed on this field was developed by Ré, Dalvi, and Suciu [92]. The main idea of their mechanism is to run one Monte-Carlo simulation in parallel for each answer candidate. So, their method only refines probabilities that are really needed to determine the top-k answers.
 - A more recent approach from Olteanu and Wen is based on shared lineage factors identified through share plans [84].
 - Dylla, Miliaraki, and Theobald published in [32] another remarkable technique for computing top-k answer sets. They designed an exact top-k pruning algorithm without materializing all answer candidates. Therefore, conjunctive queries over multiple levels of select-project-join views are evaluated before they are eventually mapped to Datalog rules. Those rules can then be directly grounded at query processing time.
 - We have also proposed a filter mechanism for determining the top-k answers of a query [78]. It is worthwhile to mention that we already used a premature version of tuple event patterns giving us more compact lineage representations within the underlying relational database layer.
- **Scoring functions:** Another significant class of probabilistic database systems combines possible-worlds-semantics with scoring functions. In principle, adequate scoring functions can express similarity and proximity conditions as *price as low as possible* and *location close to*. Such conditions can be evaluated on classical and probabilistic databases.

- The combination of possible-worlds-semantics and scoring functions was often defined under the top-k selection query model by using various probabilistic ranking semantics [50].
By following this model, score values are directly determined on attribute values of just one *single* probabilistic relation. In other words, there is *no* complex relational query involved.
- The approach described in [107] extends this simple query model to **SJ**-queries for building mashups of unstructured sources.
- **Logic-based scoring functions:** In our previous works such as [71, 72, 69, 67], we also improved the simple top-k selection paradigm in order to equip our probabilistic database system *ProQua* [70] with complex *logic-based scoring functions* applied on arbitrary **SPJU**-queries.
 - The development of ProQua was initiated through a powerful IR model originally devised by Schmitt [98]. He proposed a vector space model capable of embedding logic-based similarity conditions as well as Boolean predicates into the mathematical formalism known from quantum logic and mechanics. In the original work of Schmitt, all types of conditions are still applied on deterministic relational data.
 - In [71], we first transferred this theoretical model to a more pragmatic SQL-like query language called QSQL.
 - Afterwards, we developed in [72, 69, 67] a unified data and query model for logic-based scoring functions in conjunction
- **Weighting of subconditions and subqueries:** Besides logic-based scoring functions, we additionally enhanced ProQua with two weighting approaches for logical and algebra operators [67, 97].

7.6 Summary

In this chapter, we presented an overview of the most important and interesting works that have been published in a broader context of probabilistic databases. We briefly outlined early works addressing probabilistic data, and query models, incomplete databases and probabilistic graphical models. Afterwards, we provided a list of existing systems which have been successfully developed for managing and querying probabilistic databases. The most significant methods of the two main classes of approaches, i.e., extensional and intensional evaluation techniques, have been discussed as well. Finally, we mentioned further interesting approaches that do not directly belong to one of two aforementioned groups of techniques.

Part III

Motivation

In Part (III), we outline the motivation behind our new methods for lineage construction. Part (III) contains the following two chapters:

- Chapter (8): a description of three state-of-the-art approaches for lineage construction and
- Chapter (9): an overview of our vertical lineage construction concept.

To begin with, we introduce an example database, which is based on a real world data set, namely the IMDB movie database¹. It is used throughout our motivating explanations in Part (III).

Example 7.1 (Example database). *In Part (III), we use the tables Episodes, Keywords and Locations depicted in Figure (7.3) on Page (51) in order to set up a small TID database. It captures a tiny excerpt of the popular IMDB movie database. In detail,*

- *our table Episodes contains TV episodes belonging to different TV shows,*
- *our table Keywords saves keywords associated with those episodes, and*
- *our table Locations stores places where the plots of the chosen TV shows take place.*

Each tuple t in Figure (7.3) on Page (51) is annotated by an atomic tuple event e_t and its corresponding probability $\mathbf{P}(e_t)$. Since we deal with a TID database, all given atomic tuple events are presumed to be independent from each other.

Episodes					
	tv show	season	episode	\mathbf{E}	$\mathbf{P}(e_t)$
t_1	Dexter	1	1	e_1	0.3
t_2	Dexter	1	2	e_2	0.2
t_3	Dexter	3	3	e_3	0.4
t_4	Sopranos	2	4	e_4	0.5
t_5	Californication	2	2	e_5	0.1
t_6	Californication	2	3	e_6	0.3

Keywords					
	tv show	season	keyword	\mathbf{E}	$\mathbf{P}(e_t)$
t_7	Dexter	1	double life	e_7	0.2
t_8	Dexter	3	criminal	e_8	0.4
t_9	Sopranos	2	mob	e_9	0.5
t_{10}	Californication	2	writer	e_{10}	0.6

Locations				
	tv show	location	\mathbf{E}	$\mathbf{P}(e_t)$
t_{11}	Sopranos	USA	e_{11}	0.1
t_{12}	Sopranos	Italy	e_{12}	0.3
t_{13}	Californication	USA	e_{13}	0.8

Figure 7.3: Extracts of IMDB tables with atomic tuple events

¹<http://www.imdb.com/>

Chapter 8

State-of-the-art lineage construction techniques

Probabilistic database systems usually take advantage of a relational database layer, e.g., MystiQ [94], Trio [3], MayBMS/SPROUT [60, 82], SPROUT2 [37], PrDB [101], Orion [104], and ProQua [70]. Except for MystiQ and Orion, all these systems also exploit lineage formulas, which are directly built within their relational database layers. After a general outline of our motivation behind creating lineage formulas in Section (8.1), we introduce in this chapter three basic construction mechanisms used in the aforementioned systems:

- Section (8.2): classical construction of nested lineage formulas,
- Section (8.3): generation of lineage formulas in DNF and
- Section (8.4): creation of networks for representing lineage formulas.

Afterwards, we state in Section (8.5) our design goals, which are derived from the advantage and disadvantage of the described existing techniques.

To support our further discourse, we investigate a query applied on our probabilistic IMDB database introduced in Example (7.1) on Page (51).

Example 8.1 (Example query). *Figure (8.1) on Page (54) shows our example query Q asking for locations, where the story of at least one episode of a TV show has taken place. The considered episodes are required to be part of the first two seasons of a TV show. Query Q is evaluated on the probabilistic IMDB database of Example (7.1) on Page (51).*

8.1 Why do we use lineage formulas?

First, we justify our choice to use lineage formulas for query evaluation on probabilistic databases. In particular, we explain why we deal with lineage formulas instead of computing the desired answer probabilities directly.

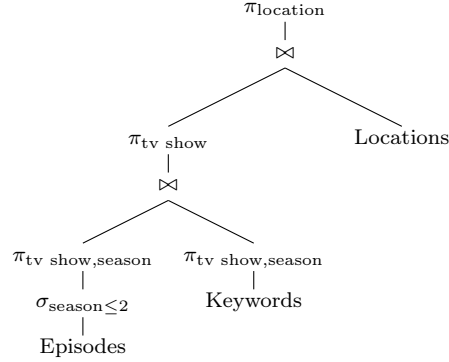
In Section (5.2), we specified the result of the evaluation of a relational algebra query Q on a probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ as follows:

$$Q(\mathbf{pdb}) = (Q_{\text{poss}(\mathcal{W})}, pr_Q) \quad \text{with} \quad \forall t \in Q_{\text{poss}(\mathcal{W})} : pr_Q(t) := \mathbf{P}(t \in Q).$$

It comprises the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$ and its corresponding answer probabilities.

To generate the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$, we can easily employ an RDBMS¹.

¹A detailed discussion is given in Chapter (13).

Figure 8.1: Example query Q

Can we also compute all answer probabilities $\mathbf{P}(t \in Q)$ by means of an RDBMS? To answer this question, we refer to the complexity of determining the answer probabilities $\mathbf{P}(t \in Q)$.

- We know from [24] that there are queries, for which calculating $\mathbf{P}(t \in Q)$ is hard for $\#\mathcal{P}$. They are called *hard queries* (Definition (5.3) on Page (31)).
- In contrast, for *tractable queries* it is possible to calculate $\mathbf{P}(t \in Q)$ in polynomial time.

In the following, we briefly elucidate both cases with regards to our question under investigation.

Evaluation of hard queries

In general, no algorithm is known that can decide whether a given query Q from **SPJUD** represents a hard or a tractable query, if we only consider the query structure of Q [111].

We cannot precisely exclude *all* hard queries from our query language. Hence, the restriction to tractable queries is in general not possible.

In other words, each algorithm for computing $\mathbf{P}(t \in Q)$ with $Q \in \mathbf{SPJUD}$ has to deal with queries, which are not evaluable in polynomial time. RDBMSs are not designed to cope this kind of problems.

All operators and data types of a standard RDBMS are usually developed and optimized to evaluate queries in polynomial time.

As a consequence, we aim to transfer the problem of computing $\mathbf{P}(t \in Q)$ from the relational database layer into an additional probabilistic query engine, where further anytime evaluation and approximation algorithms can be exploited.

In order to establish the 2-tier basic architecture of probabilistic database systems, we need a tool, which connects the generation of $Q_{\text{poss}(\mathcal{W})}$ within the relational database layer and the computation of $\mathbf{P}(t \in Q)$ performed outside of an RDBMS.

Lineage formulas, which are capable of expressing tuple events of the queried probabilistic database can be employed very well for this task.

Classical lineage construction for Q	
location	φ^t
USA	$((e_4 \wedge e_9) \wedge e_{11}) \oslash (((e_5 \oslash e_6) \wedge e_{10}) \wedge e_{13})$
Italy	$(e_4 \wedge e_9) \wedge e_{12}$

Figure 8.2: SPROUT2: Query results for Q using classical lineage construction

Evaluation of tractable queries

Previous works such as [24, 26] proved that tractable queries of several query subclasses of **SPJUD** can be evaluated directly via an RDBMS. However, Olteanu et al. showed in [82] that the computation of $\mathbf{P}(t \in Q)$ for tractable queries should be also executed within a dedicated probabilistic query engine.

The direct computation of $\mathbf{P}(t \in Q)$ within an RDBMS usually leads to very inefficient relational query plans [82].

Following [82], we always separate the probability computation task from the relational data processing part and use lineage formulas as an effective tool for bridging both components.

8.2 Classical construction of nested lineage formulas

The most obvious way to build lineage formulas is to apply the classical construction rules of Fuhr and Röllecke presented in Lemma (6.1) on Page (35). By using these rules, Fink, Olteanu, and Rath proposed to generate nested lineage formulas within an RDBMS directly [37].

In their framework SPROUT2, each row of a relation is extended by an additional attribute field containing a constructed lineage formula.

Example 8.2 (Classical lineage construction (Fuhr and Röllecke, SPROUT2)). *The table of Figure (8.2) on Page (55) contains the two determined answer tuples for Q :*

$$Q_{\text{poss}(\mathcal{W})} = \{(USA)_{\text{location}}, (Italy)\},$$

which are augmented by their lineage formulas:

$$\begin{aligned}\varphi_Q^{USA} &= ((e_4 \wedge e_9) \wedge e_{11}) \oslash (((e_5 \oslash e_6) \wedge e_{10}) \wedge e_{13}) \\ \varphi_Q^{Italy} &= (e_4 \wedge e_9) \wedge e_{12}.\end{aligned}$$

Please remember that lineage formulas can grow polynomially, if we take the number of all possible tuples into account, see Lemma (6.2) on Page (37). This means that an attribute field storing a single lineage formula could exceed the entire input database. Lineage formulas with a size of 14 MB for a single tuple have already been observed in practical applications [93]. Most critically, the processing of very large attribute fields can slow down an RDBMS tremendously, see our experiments in Chapter (13).

8.3 Generation of lineage formulas in DNF

As an alternative to a nested lineage formula constructed in a single attribute field, we could use U-relations devised by Antova, Jansen, Koch, and Olteanu for the MayBMS system [7].

U-relation for Q			
location	$\mathbf{E}_{Episodes}$	$\mathbf{E}_{Locations}$	$\mathbf{E}_{Keywords}$
USA	e_9	e_4	e_{11}
USA	e_{10}	e_5	e_{13}
USA	e_{10}	e_6	e_{13}
Italy	e_9	e_4	e_{12}

Figure 8.3: MayBMS: U-relation for Q

The key idea of U-databases is a fast and simple construction of lineage formulas in DNF. Since lineage formulas in DNF are very well-structured, they can be easily created and stored by relational standard operators and data types.

Example 8.3 (Lineage construction in DNF (MayBMS/SPROUT)). *In Figure (8.3) on Page (56), we show the U-relation generated for our example query Q . Because each row of a U-relation stores exactly one conjunct, we need to disjunctively combine all conjuncts of rows, which have the same value for the attribute *location* in order to achieve a specific final lineage formula.*

Thus, the first three conjuncts of Figure (8.3) on Page (56) form φ_Q^{USA} :

$$\varphi_Q^{USA} = (e_9 \oslash e_4 \oslash e_{11}) \vee (e_{10} \oslash e_5 \oslash e_{13}) \vee (e_{10} \oslash e_6 \oslash e_{13}).$$

As already seen in Section (6.3), the transformation of a lineage formula into a DNF can be very expensive. In particular, lineage formulas, which involve negated subformulas can grow dramatically, if we unfold them into DNF.

MayBMS avoids this problem by strictly forbidding difference operations within its query language. However, the importance and indispensability of providing full algebra support for a probabilistic query language have been strongly underlined by several significant publications in this field [36, 57, 37].

8.4 Creation of networks for representing lineage formulas

Our next basic approach sets up networks, which can be easily used to encode lineage formulas. It comprises two prominent representatives, namely the PrDB and the Trio system.

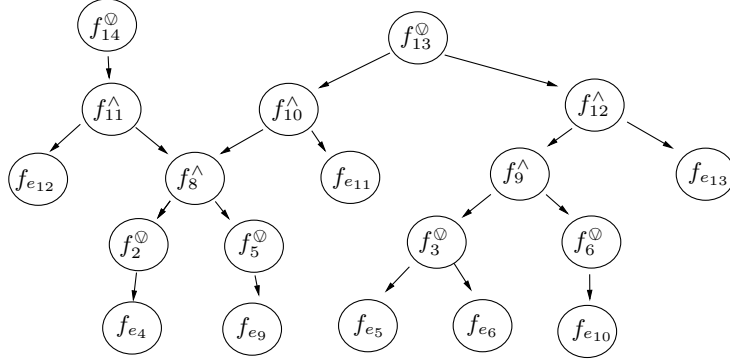
Networks of factors in PrDB

In [101], Sen, Deshpande, and Getoor presented several interesting techniques aimed at empowering their probabilistic database system PrDB. They extensively used probabilistic graphical models (PGM) in order to express rich correlations among tuple events.

To be more specific, PrDB sets up networks consisting of recursively defined random variables. Such random variables are called *factors*. Thereby, a specific factor $f_{id}^{type}(op_1, \dots, op_n)$ stands for one lineage *subformula*. It is described by the following three values:

- a unique number id assigned during the construction process,
- a factor type $type$ derived from the top-most logical operation of the corresponding lineage subformula, and
- a list of factor operands op_1, \dots, op_n .

The emerging networks can then be easily interpreted as lineage formulas.

Figure 8.4: PrDB: Factor network representing φ_Q^{USA} and φ_Q^{Italy}

Example 8.4 (Factor network (PrDB)). We exemplify the correspondences between factor networks and lineage formulas by means of the network that encodes our lineage formulas φ_Q^{USA} and φ_Q^{Italy} . It is illustrated in Figure (8.4) on Page (57).

In this network, each node expresses a lineage subformula of φ_Q^{USA} and φ_Q^{Italy} . To resemble a specific lineage subformula, we just need to follow the respective operand references.

Moreover, when we read all logical operators of φ_Q^{USA} and φ_Q^{Italy} in prefix notation, we can easily recover the structures of our lineage formulas, i.e.,

$$\begin{aligned}\varphi_Q^{USA} &= ((e_4 \wedge e_9) \wedge e_{11}) \odot (((e_5 \odot e_6) \wedge e_{10}) \wedge e_{13}) \\ &\equiv \odot(\wedge(\wedge(\odot(e_4), \odot(e_9)), e_{11}), \wedge(\wedge(\odot(e_5, e_6), \odot(e_{10})), e_{13})) \\ \varphi_Q^{Italy} &= (e_4 \wedge e_9) \wedge e_{12} \\ &\equiv \odot(\wedge(\wedge(\odot(e_4), \odot(e_9)), e_{12}))\end{aligned}$$

are encoded by

$$f_{13}^{\odot}(f_{10}^{\wedge}(f_8^{\wedge}(f_2^{\odot}(f_{e_4}), f_5^{\odot}(f_{e_9})), f_{e_{11}}), f_{12}^{\wedge}(f_9^{\wedge}(f_3^{\odot}(f_{e_5}, f_{e_6}), f_6^{\odot}(f_{e_{10}})), f_{e_{13}})),$$

and

$$f_{14}^{\odot}(f_{11}^{\wedge}(f_8^{\wedge}(f_2^{\odot}(f_{e_4}), f_5^{\odot}(f_{e_9})), f_{e_{12}})).$$

PrDB creates its factors with rules, which are basically derived from the classical construction rules of Lemma (6.1) on Page (35). In contrast to SPROUT2 and MayBMS, PrDB does not create its encoded lineage formulas within one resulting relation. Instead, it stores its built factors and their references to each other in a *set of relations* that directly correspond to the subqueries of the given input query.

Example 8.5 (Tables of factors (PrDB)). Figure (8.5) on Page (58) depicts the five relations containing the factors of our example network of Figure (8.4) on Page (57). Each relation relies on one of the following five subqueries of Q :

$$\begin{aligned}sq_1 &:= \pi_{tv \ show, \ season}(\sigma_{season \leq 2}(Episodes)) \\ sq_2 &:= \pi_{tv \ show, \ season}(Keywords) \\ sq_3 &:= \pi_{tv \ show}(sq_1 \bowtie sq_2) \\ sq_4 &:= \pi_{tv \ show}(sq_1 \bowtie sq_2) \bowtie Locations\end{aligned}$$

Factors of sq_1			Factors of sq_2		
tv show	season	factor	tv show	season	factor
Dexter	1	$f_1^\odot(f_{e_1}, f_{e_2})$	Dexter	1	$f_4^\odot(f_{e_7}, f_{e_8})$
Sopranos	2	$f_2^\odot(f_{e_4})$	Sopranos	2	$f_5^\odot(f_{e_9})$
Californication	2	$f_3^\odot(f_{e_5}, f_{e_6})$	Californication	2	$f_6^\odot(f_{e_{10}})$

Factors of sq_3	
tv show	factor
Dexter	$f_7^\wedge(f_1^\odot, f_4^\odot)$
Sopranos	$f_8^\wedge(f_2^\odot, f_5^\odot)$
Californication	$f_9^\wedge(f_3^\odot, f_6^\odot)$

Factors of sq_4			Factors of sq_5	
tv show	location	factor	location	factor
Sopranos	USA	$f_{10}^\wedge(f_8^\wedge, f_{e_{11}})$	USA	$f_{13}^\odot(f_{10}^\wedge, f_{12}^\wedge)$
Sopranos	Italy	$f_{11}^\wedge(f_8^\wedge, f_{e_{12}})$	Italy	$f_{14}^\odot(f_{11}^\wedge)$
Californication	USA	$f_{12}^\wedge(f_9^\wedge, f_{e_{13}})$		

Figure 8.5: PrDB: Introduced factors for Q

$$sq_5 := \pi_{location}(sq_4).$$

In contrast to Figure (6.1) on Page (36), the relations in Figure (8.5) on Page (58) do not represent volatile results of intermediate processing steps. They rather embody the final outcome produced by the relational database layer of PrDB. Accordingly, Figure (8.5) on Page (58) shows a distribution of 14 tuples over five relations. In comparison, the corresponding U-relation in Figure (8.3) on Page (56) only contains four tuples.

As mentioned before, PrDB adds one factor for each lineage subformula occurred during the query evaluation process. This implies that PrDB also introduces a large number of unnecessary factors. Those factors embody intermediate lineage subformulas, which do not contribute to the final lineage formulas.

Example 8.6 (Unnecessary introduced factors (PrDB)). In Example (8.5) on Page (57), we can find f_1^\odot , f_4^\odot , and f_7^\wedge as factors added for sq_1 , sq_2 , and sq_3 . They are neither part of φ_Q^{USA} nor φ_Q^{Italy} , see Example (8.4).

Networks of Boolean functions in Trio

Trio is another interesting probabilistic database system that intensively relies on lineage formulas [11, 28]. Benjelloun, Das Sarma, Halevy, Theobald, and Widom developed *Uncertainty-Lineage Databases* (ULDBs) as the key concept for Trio. From a construction perspective, ULDBs make use of a technique, which is similar to the one applied for factor networks of PrDB. They also manage references to lineage subformulas explicitly within a set of relations.

Instead of factor networks, Trio defines networks of Boolean functions. To highlight the similarities between Trio and PrDB, we address a specific Boolean function by an identifier $\lambda_{id}^{type}(op_1, \dots, op_n)$ that has the same components as a factor identifier, see above.

Example 8.7 (Network of recursive Boolean functions (Trio)). *Analogously to the nested infix notation of factor networks (see Example (8.4) on Page (57)), we can express the structures of our lineage formulas*

$$\varphi_Q^{USA} = ((e_4 \wedge e_9) \wedge e_{11}) \odot (((e_5 \odot e_6) \wedge e_{10}) \wedge e_{13}) \quad \text{and} \quad \varphi_Q^{Italy} = (e_4 \wedge e_9) \wedge e_{12}$$

as recursively formulated Boolean functions:

$$\lambda_{13}^{\odot}(\lambda_{10}^{\wedge}(\lambda_8^{\wedge}(\lambda_2^{\odot}(\lambda_{e_4}), \lambda_5^{\odot}(\lambda_{e_9})), \lambda_{e_{11}}), \lambda_{12}^{\wedge}(\lambda_9^{\wedge}(\lambda_3^{\odot}(\lambda_{e_5}, \lambda_{e_6}), \lambda_6^{\odot}(\lambda_{e_{10}})), \lambda_{e_{13}}))$$

and

$$\lambda_{14}^{\odot}(\lambda_{11}^{\wedge}(\lambda_8^{\wedge}(\lambda_2^{\odot}(\lambda_{e_4}), \lambda_5^{\odot}(\lambda_{e_9})), \lambda_{e_{12}})).$$

In contrast to PrDB, the relational database layer of Trio generates two types of resulting relations. Benjelloun et al. proposed to use a set of table pairs containing the relational data part and the references to all lineage subformulas separated from each other.

Example 8.8 (Data and lineage tables (Trio)). *Figure (8.6) on Page (60) depicts a simplified form of the data and lineage tables created for our example query Q . Both types of relations are connected by the identifiers introduced for the respective Boolean functions (attribute `lid`). Thereby, each row of a specific lineage table contains a single operand (attribute `operand-source` and attribute `operand-lid`) of a Boolean function (attribute `lid`).*

Despite Trio proposes several minimization operations to remove obsolete Boolean functions [11], we can conclude that the demonstrated networks of PrDB and Trio follow the same idea in terms of lineage construction. In both cases, we need to create at least one tuple per final lineage subformula. In Lemma (6.2) on Page (37), we already gave the maximal length for a lineage formula. Accordingly, the sizes of the produced result sets are bounded by

$$|Q| * \mathbf{max}(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|},$$

which is more we are used from the deterministic query evaluation (i.e., $\mathbf{max}(|R_1|, \dots, |R_m|)^{|Q|}$).

8.5 Design goals for lineage construction

In the previous sections, we presented existing approaches for lineage construction. By taking their positive and negative properties into account, we state the following four design goals for our approach:

- Design goal (1): the query language allows all relational algebra operators,
- Design goal (2): the length of a lineage formula is not limited,
- Design goal (3): all used relational operations and data types can be natively implemented,
- Design goal (4): the relational result sizes are asymptotically not greater than known from the deterministic case.

All given criteria strive for our ultimate goal to obtain a probabilistic query evaluation that would be similar to the one we use for the deterministic case.

Next, we briefly assess the presented basic approaches with regard to our four established design goals. In particular, we underline their main aspects that fail our goals.

Data of sq_1			Lineage of sq_1		
tv show	season	lid	lid	operand-source	operand-lid
Dexter	1	λ_1°	λ_1°	Episodes	λ_{e_1}
Sopranos	2	λ_2°	λ_1°	Episodes	λ_{e_2}
Californication	3	λ_3°	λ_2°	Episodes	λ_{e_4}
			λ_3°	Episodes	λ_{e_5}
			λ_3°	Episodes	λ_{e_6}

Data of sq_2			Lineage of sq_2		
tv show	season	lid	lid	operand-source	operand-lid
Dexter	1	λ_4°	λ_4°	Keywords	λ_{e_7}
Sopranos	2	λ_5°	λ_4°	Keywords	λ_{e_8}
Californication	2	λ_6°	λ_5°	Keywords	λ_{e_9}
			λ_6°	Keywords	$\lambda_{e_{10}}$

Data of sq_3		Lineage of sq_3		
tv show	lid	lid	operand-source	operand-lid
Dexter	λ_7^\wedge	λ_7^\wedge	sq_1	λ_1°
Sopranos	λ_8^\wedge	λ_7^\wedge	sq_2	λ_1°
Californication	λ_9^\wedge	λ_8^\wedge	sq_1	λ_2°
		λ_8^\wedge	sq_2	λ_5°
		λ_9^\wedge	sq_1	λ_3°
		λ_9^\wedge	sq_2	λ_6°

Data of sq_4			Lineage of sq_4		
tv show	location	lid	lid	operand-source	operand-lid
Sopranos	USA	λ_{10}^\wedge	λ_{10}^\wedge	sq_3	λ_8^\wedge
Sopranos	Italy	λ_{11}^\wedge	λ_{10}^\wedge	Locations	$\lambda_{e_{11}}$
Californication	USA	λ_{12}^\wedge	λ_{11}^\wedge	sq_3	λ_8^\wedge
			λ_{11}^\wedge	Locations	$\lambda_{e_{12}}$
			λ_{12}^\wedge	sq_3	λ_9°
			λ_{12}^\wedge	Locations	$\lambda_{e_{13}}$

Data of sq_5		Lineage of sq_5		
location	lid	lid	operand-source	operand-lid
USA	λ_{13}°	λ_{13}°	sq_4	λ_{10}^\wedge
Italy	λ_{14}°	λ_{13}°	sq_4	λ_{12}^\wedge
		λ_{14}°	sq_4	λ_{11}^\wedge

Figure 8.6: Trio: Generated data and lineage tables for Q

Main issues of classical construction rules (Fuhr and Röllecke, SPROUT2)

At first sight, it seems that Fuhr and Röllecke have already provided us with everything we need. In Section (6.2) and (8.2), we described and exemplified their construction rules incorporating all relational operators.

The main problem of the classical construction rules is their practical implementation. Most critically, there is no support of logical formulas in the form of native data types and operators through a standard RDBMS.

The direct construction of nested lineage formulas within an RDBMS is very inefficient, since nested lineage formulas cannot be natively represented and processed.

In principle, we have to encode the irregular form of a logical formula by means of a user-defined data type or a large string, e.g., in JSON format. This does not fit our Design Goal (3).

In addition, we already proved in Lemma (6.2) that the length of a single lineage formula grows polynomially in the size of database.

One attribute field of *one* resulting row can already exceed the size of the input database.

If we take into consideration that each attribute field in a RDBMS has a limited size, we have to restrict the length of a lineage formula in advance. This obviously violates our Design goal (2).

Main issues of lineage formulas in DNF (MayBMS)

Instead of dealing with the irregular forms of nested lineage formulas, we can work with lineage formulas in DNF. U-relations of the MayBMS system embody such an approach, see Section (8.3). Since they store each conjunct in a single row, lineage formulas in DNF are stored *horizontally* within the resulting relation.

However, we already know from Section (6.3) that the total number of conjuncts of a lineage formula in DNF can explode. In fact, a table, which stores one conjunct per row can grow *exponentially* considering the size of the database.

In practice, the construction of arbitrary lineage formulas in DNF is intractable.

To the best of our knowledge, all techniques based on DNF avoid this problem by completely forbidding the difference operator in their query languages. This obviously violates our Design goal (1).

Restricted lineage formulas in DNF exclude relational standard operators from their query languages.

Fulfillment of design goals for lineage construction					
	design goal	safe plans (MystiQ)	nested (SPROUT2)	DNF (MayBMS)	networks (PrDB/Trio)
1	full relational algebra support	no	yes	no	yes
2	unlimited lineage formula lengths	-	no	yes	yes
3	native relational data types	yes	no	yes	yes
4	result set sizes as in the deterministic case	yes	yes	yes	no

Figure 8.7: Fulfillment of design goals for lineage construction

Main issues of factor/Boolean function networks (PrDB/Trio)

Section (8.4) sketched networks of factors and Boolean functions as our final basic technique. These networks can be easily interpreted as lineage formulas.

Generally speaking, they store references to each lineage subformula within a *set* of relations. Thereby, each of their relations is associated with a *subquery* of the given input query, see Figure (8.5) and (8.6) on Page (58) and (60).

In total, the relational database layer of PrDB/Trio produces up to

$$|Q| * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$$

resulting tuples.

The sizes of the relational results of PrDB/Trio do not satisfy our Design goal (4).

Besides the negative effect of large result sets for the RDBMS query processing part, we also point out the implications for the probabilistic query engine. Obviously, the outcome of the relational database layer directly correlates to the input size of the probabilistic query engine.

Large input sizes for the probabilistic query engine can significantly slow down the overall query processing.

8.6 Summary

Figure (8.7) on Page (62) shows our final matrix illustrating the fulfillment of the stated design goals. In fact, none of the discussed basic approaches is capable of satisfying all required design goals. This conclusion motivated us to look for alternative lineage construction methods.

Chapter 9

Vertical lineage construction in a nutshell

In this chapter, we provide an overview of our novel lineage construction approach. In particular, we highlight our key ideas in a very focused and compact manner. The missing theoretical and technical details are laid out in the following parts of our work. The following topics are covered:

- Section (9.1): our adjusted basic architecture,
- Section (9.2): our theoretical framework, and
- Section (9.3): our main construction algorithm.

9.1 Adjusted basic architecture

First of all, we repeat in Figure (9.1) on Page (64) the basic architecture of a traditional probabilistic database system that exploits lineage formulas. The schema of Figure (9.1) on Page (64) illustrates how existing systems basically consist of a relational database layer and a probabilistic query engine. Thereby, lineage formulas are firstly generated within the used RDBMS and subsequently transferred to a second-storage query engine in order to optimize and evaluate them.

In the last chapter, we already identified serious drawbacks for a system relying on this basic architecture, see Figure (8.7) on Page (62). In essence, we conclude following two learnings from the existing state-of-the-art systems:

Lineage formulas have to be constructed in a *nested form* in order to provide full relational algebra support.

and

Nested lineage formulas cannot be *efficiently* generated within an RDBMS, since they are not natively supported by a standard RDBMS.

Consequently, we propose to separate the creation of the irregular forms of nested lineage formulas from the generation of the relational result part. That paradigm shift is illustrated in Figure (9.2) on Page (64).

The actual construction of nested lineage formulas is not performed within our relational database layer. Instead, we construct all lineage formulas by our probabilistic query engine.

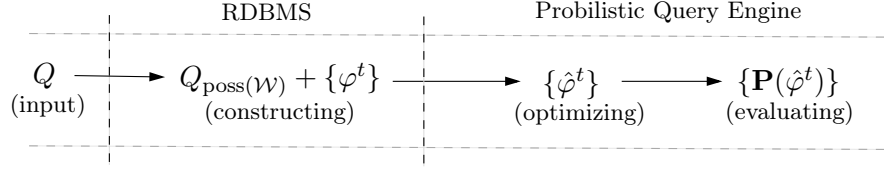


Figure 9.1: Traditional architecture of a probabilistic database system

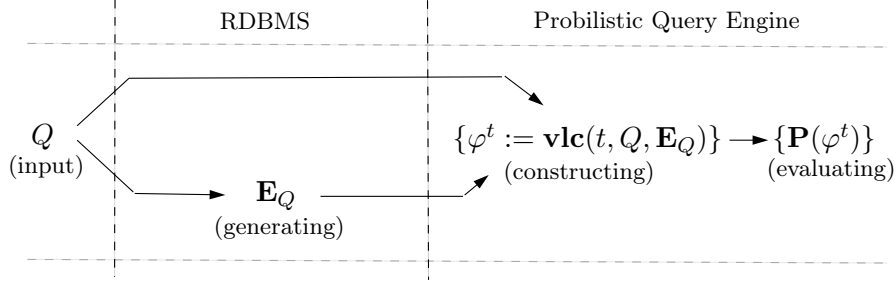


Figure 9.2: Adjusted architecture of Prophecy with vertical lineage construction

Please note that traditional approaches also (re)-create lineage formulas in their second-storage components in order to optimize and evaluate them. However, the structures of those lineage formula are already generated in the RDBMS. They need to be transferred from the relational database layer to the probabilistic query engine.

In order to demonstrate our following main concepts, we examine a query applied on our three IMDB scenario tables given in Example (7.1) on Page (51).

Example 9.1 (Example query). *In Chapter (9), we consider the query*

$$Q \equiv \pi_{(tv\ show, \ season)}(Episodes) \setminus \pi_{(tv\ show, \ season)}(Keywords \bowtie Location)$$

on the probabilistic database of Example (7.1) on Page (51). It asks for the first seasons of all TV shows for which no keywords and locations are stored for. Using our query semantics of Definition (5.2) on Page (30), we can determine two answer tuples for Q :

$$Q_{poss(W)} = \{(\underline{Dexter}, 1)_{(tv\ show, \ season)}, (\underline{Dexter}, 3)\} = \{(Dex, 1)_{(tv\ show, \ season)}, (Dex, 3)\}.$$

9.2 Overview of theoretical framework

In Section (6.2), we already discussed how the classical rules can be used for determining lineage formulas. By applying them, we achieve the following two lineage formulas for our two answer tuples:

$$\begin{aligned} \varphi^{(Dex,1)} &= (e_1 \odot e_2) \wedge \neg F \\ \varphi^{(Dex,3)} &= e_3 \wedge \neg F. \end{aligned}$$

The rules of Lemma (6.1) on Page (35) first and foremost specify one possible form of $\varphi^{(Dex,1)}$ and $\varphi^{(Dex,3)}$. We still have to find an efficient way of constructing $\varphi^{(Dex,1)}$ and $\varphi^{(Dex,3)}$ within a probabilistic database system. The following discourse presents a novel approach for this task.

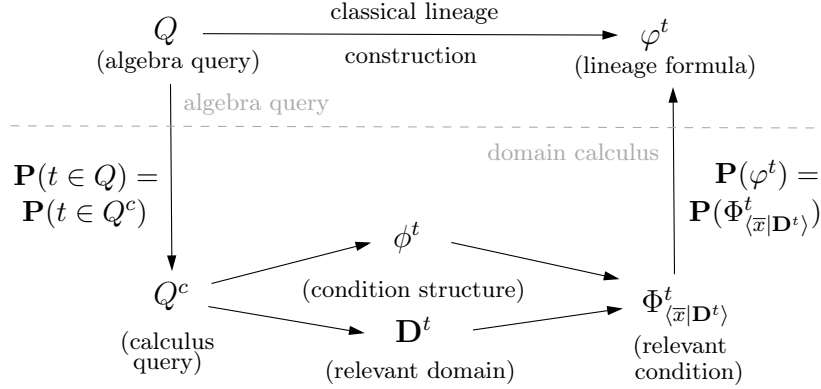


Figure 9.3: Conceptual transformation chain

Thereby, we distill our main ideas without going into too much detail. The following chapters explain and verify all mentioned techniques and methods.

Transformation chain

The backbone of our theoretical framework is the conceptual transformation chain depicted in Figure (9.3) on Page (65). It consists of the following three main steps:

- Main step (1): considering the input query Q as an equivalent domain calculus query Q^c ,
- Main step (2): building relevant conditions $\Phi^t_{\langle \bar{x} | \mathbf{D}^t \rangle}$ over relevant domains \mathbf{D}^t , and
- Main step (3): mapping relevant conditions to lineage formulas φ^t .

We emphasize that the transformation steps of Figure (9.3) on Page (65) are purely *conceptual*. Our pragmatic algorithm does *not* need to execute them. Their main purpose is rather to prove the equivalence between the classical and our alternative constructed lineage formulas.

The transformation chain of Figure (9.3) on Page (65) is not practically implemented.

Main step (1): Considering the input query as domain calculus query

To begin with, we reformulate our input query into an equivalent query of the relational domain calculus.

In order to address formula structures and domains independently, we exploit domain calculus queries.

In principle, each algebra query can be expressed as an equivalent domain calculus query [74]. In Lemma (10.1) on Page (84), we present a set of mapping rules between relational algebra and domain calculus operators. By using these rules and the labels of Figure (9.4), we can rewrite our example query

$$Q = \pi_{(\text{tv show, season})}(\text{Episodes}) \setminus \pi_{(\text{tv show, season})}(\text{Keywords} \bowtie \text{Location})$$

Short forms and variables	
relation/attribute	relation/variable
Episodes	R_{epi}
Keywords	R_{key}
Locations	R_{loc}
tv show	x_t
season	x_s
episode	x_e
keyword	x_k
location	x_l

Figure 9.4: Short forms and variables for IMDB tables

into

$$Q^c = \left\{ \underbrace{(x_t, x_s)}_t \mid \underbrace{\exists x_e : R_{epi}(x_t, x_s, x_e) \wedge \neg(\exists x_k, x_l : (R_{key}(x_t, x_s, x_k) \wedge R_{loc}(x_t, x_l)))}_{\Phi(x_t, x_s)} \right\}.$$

Roughly speaking¹, our domain calculus query Q^c evaluates its first-order *condition* $\Phi(x_t, x_s)$ over a set of *infinite variable domains*:

$$\begin{aligned} \mathbf{D}_{x_t} &= \{\text{Dexter, Sopranos, Californication}, \dots\}, \\ \mathbf{D}_{x_s} &= \mathbf{D}_{x_e} = \{1, 2, \dots\} \quad \text{and} \\ \mathbf{D}_{x_k} &= \{\text{double life, crime, mob}, \dots\}. \end{aligned}$$

When the condition $\Phi(x_t, x_s)$ is satisfied by a domain value $(x_t, x_s) \in (\mathbf{D}_{x_t} \times \mathbf{D}_{x_s})$, this domain value forms an answer tuple $t = (x_t, x_s)$. The set of all resulting tuples determines the query result of Q^c .

In Chapter (10), we tailor the traditional form of the domain calculus towards our purposes. Specifically, we consider for each answer tuple t an explicit condition Φ^t .

The conditions Φ^t are transformed stepwisely into our alternative lineage formulas φ^t of Figure (9.3) on Page (65).

To set up a specific condition Φ^t , we directly write the values of an answer tuple $t = (x_t, x_s)$ into the general condition Φ . By doing so, we achieve the following two conditions for our two answer tuples $(\text{Dex}, 1)_{(x_t, x_s)}$ and $(\text{Dex}, 3)_{(x_t, x_s)}$ of Example (9.1) on Page (64):

$$\begin{aligned} \Phi^{(\text{Dex}, 1)} &= \exists x_e : R_{epi}(\text{Dex}, 1, x_e) \wedge \neg(\exists x_k, x_l : (R_{key}(\text{Dex}, 1, x_k) \wedge R_{loc}(\text{Dex}, x_l))) \\ \Phi^{(\text{Dex}, 3)} &= \exists x_e : R_{epi}(\text{Dex}, 3, x_e) \wedge \neg(\exists x_k, x_l : (R_{key}(\text{Dex}, 3, x_k) \wedge R_{loc}(\text{Dex}, x_l))). \end{aligned}$$

Main step (2): Building relevant conditions over relevant domains

Our second transformation step consists of

- resolving of all quantifiers,
- separation of formula structures from domains, and
- setting up of relevant conditions.

¹A more precise definition of the relational domain calculus is given in Chapter (10).

Resolving of all quantifiers

In the next step, we equivalently replace all universal and existential quantifiers with our n-ary operators \bigcirc and \bigodot .

The substitution of all quantifiers in Φ^t brings our conditions closer to the target formalism, i.e., lineage formulas formulated as propositional formulas. Moreover, it directly encodes the underlying domains into our conditions.

Thus, we obtain (infinite) propositional formulas denoted as $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$, where n-ary operators explicitly range over their corresponding variable domains, e.g.,

$$\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^{(\text{Dex}, 1)} = \bigodot_{x_e \in \mathbf{D}_{x_e}} R_{\text{epi}}(\text{Dex}, 1, x_e) \wedge \neg \left(\bigodot_{x_k \in \mathbf{D}_{x_k}, x_l \in \mathbf{D}_{x_l}} R_{\text{key}}(\text{Dex}, 1, x_k) \wedge R_{\text{loc}}(\text{Dex}, x_l) \right).$$

Separating formula structures from domains

On the basis of $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$, we next isolate the domain specific and unspecific parts of our conditions by exploiting the transformation rules known from the prenex normal form (PNF).

More concretely, we equivalently rewrite Φ^t into a form, where all n-ary operators (i.e., all former quantifiers) are located at the beginning, e.g.,

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^{(\text{Dex}, 1)} &= \bigodot_{x_e \in \mathbf{D}_{x_e}} R_{\text{epi}}(\text{Dex}, 1, x_e) \wedge \neg \left(\bigodot_{x_k \in \mathbf{D}_{x_k}, x_l \in \mathbf{D}_{x_l}} R_{\text{key}}(\text{Dex}, 1, x_k) \wedge R_{\text{loc}}(\text{Dex}, x_l) \right) \\ &\equiv \underbrace{\bigodot_{x_e \in \mathbf{D}_{x_e}} \left(\bigodot_{x_k \in \mathbf{D}_{x_k}, x_l \in \mathbf{D}_{x_l}} \right)}_{\text{domain specific}} \underbrace{\left(R_{\text{epi}}(\text{Dex}, 1, x_e) \wedge \neg (R_{\text{key}}(\text{Dex}, 1, x_k) \wedge R_{\text{loc}}(\text{Dex}, x_l)) \right)}_{\text{domain unspecific}}. \end{aligned}$$

In order to address the domain specific and unspecific part more precisely, we introduce two useful notations. First, all n-ary operators are summarized in a sequence of n-ary operators:

$$\overline{\bigodot \bigodot}_{(x_e, x_k, x_l) \in \mathbf{D}} (\dots) := \bigodot_{x_e \in \mathbf{D}_{x_e}} \left(\bigodot_{x_k \in \mathbf{D}_{x_k}, x_l \in \mathbf{D}_{x_l}} (\dots) \right) \quad \text{with} \quad \mathbf{D} = (\mathbf{D}_{x_e} \times \mathbf{D}_{x_k} \times \mathbf{D}_{x_l}).$$

Following this, we denote the remaining part of Φ^t as ϕ^t :

$$\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \overline{\bigodot \bigodot}_{(x_e, x_k, x_l) \in \mathbf{D}} \phi^t(x_e, x_k, x_l),$$

where

$$\phi^t(x_e, x_k, x_l) := R_{\text{epi}}(t_{x_t}, t_{x_s}, x_e) \wedge \neg (R_{\text{key}}(t_{x_t}, t_{x_s}, x_k) \wedge R_{\text{loc}}(t_{x_t}, x_l)).$$

We consider ϕ^t as the *condition structure* of Φ^t , since it captures the same logical combination of atomic predicates for all domain values of \mathbf{D} .

Setting up relevant conditions

In the previous section, we transformed our general condition Φ^t into a form, where the condition structure ϕ^t and the underlying domain \mathbf{D} are separated. But we still deal with *infinite* propositional formulas inferred from one general domain \mathbf{D} .

In order to obtain domains that can be efficiently computed by an RDBMS, we propose our concept of *relevant domains* \mathbf{D}^t .

In short, a relevant domain $\mathbf{D}^t \subseteq \mathbf{D}$ only includes domain values d for which at least one world $W \in \mathcal{W}$ exists, for which the condition structure $\phi^t(d)$ can be fulfilled. Consequently, we omit all domain values d that are not capable of satisfying the condition structure $\phi^t(d)$ in any world. Instead of one general domain \mathbf{D} , we then work with a specific relevant domain \mathbf{D}^t for each condition Φ^t . At this point², we directly provide the required relevant domains for our example:

$$\begin{aligned} \mathbf{D}^{(\text{Dex},1)} &= \{(1)_{x_e}, (2)\} \times \mathbf{D}_{x_k} \times \mathbf{D}_{x_l} \\ &= \{(1)_{x_e}, (2)\} \times \{(_)_{x_k}\} \times \{(_)_{x_l}\} \\ &= \{(1, _, _)_{(x_e, x_k, x_l)}, (2, _, _)_{(x_e, x_k, x_l)}\} \\ \mathbf{D}^{(\text{Dex},3)} &= \{(3)_{x_e}\} \times \mathbf{D}_{x_k} \times \mathbf{D}_{x_l} \\ &= \{(3)_{x_e}\} \times \{(_)_{x_k}\} \times \{(_)_{x_l}\} \\ &= \{(3, _, _)_{(x_e, x_k, x_l)}\}. \end{aligned}$$

Please note that there are variables, for which their relevant domain values can range over their entire domains, e.g., x_k and x_l . We indicate such variables with the wildcard symbol $_$.

Taking advantage of our newly introduced relevant domains \mathbf{D}^t , we can set up our final *relevant conditions* denoted as $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$:

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t := \bigvee_{(x_e, x_k, x_l) \in \mathbf{D}^t} \phi^t(x_e, x_k, x_l)$$

leading to

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},1)} &= R_{\text{epi}}(\text{Dex}, 1, 1) \wedge \neg(R_{\text{key}}(\text{Dex}, 1, _) \wedge R_{\text{loc}}(\text{Dex}, _)) \vee \\ &\quad R_{\text{epi}}(\text{Dex}, 1, 2) \wedge \neg(R_{\text{key}}(\text{Dex}, 1, _) \wedge R_{\text{loc}}(\text{Dex}, _)) \\ \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},3)} &= R_{\text{epi}}(\text{Dex}, 3, 3) \wedge \neg(R_{\text{key}}(\text{Dex}, 3, _) \wedge R_{\text{loc}}(\text{Dex}, _)). \end{aligned}$$

Lastly, we can revert the rewriting caused by our PNF step to achieve a more compact form:

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},1)} &= (R_{\text{epi}}(\text{Dex}, 1, 1) \vee R_{\text{epi}}(\text{Dex}, 1, 2)) \wedge \neg(R_{\text{key}}(\text{Dex}, 1, _) \wedge R_{\text{loc}}(\text{Dex}, _)) \\ \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},3)} &= R_{\text{epi}}(\text{Dex}, 3, 3) \wedge \neg(R_{\text{key}}(\text{Dex}, 3, _) \wedge R_{\text{loc}}(\text{Dex}, _)). \end{aligned}$$

Main step (3): Mapping to lineage formulas

In the last part of our transformation chain, we map relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ to our final alternative lineage formulas φ^t .

For this purpose, we first introduce a binary random variable over our considered probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ for each relation predicate occurring in $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$:

$$X_{R(\bar{c})}(W) := \begin{cases} \text{T} & \text{if } (\bar{c}) \in R(W) \\ \text{F} & \text{else.} \end{cases}$$

These random variables can be used to describe our already known atomic tuple events of Definition (6.2) on Page (34):

$$\{W \in \mathcal{W} \mid X_{R(\bar{c})}(W)\} = \{W \in \mathcal{W} \mid (\bar{c}) \in R(W)\}.$$

For instance, the atomic tuple event e_1 , which is associated with the tuple $t_1 = (\text{Dex}, 1, 1)$ of table *Episodes* (Figure (7.3) on Page (51)) is then described by

$$e_1 = \{W \in \mathcal{W} \mid X_{R_{\text{epi}}(\text{Dex}, 1, 1)}(W)\} = \{W \in \mathcal{W} \mid (\text{Dex}, 1, 1) \in R_{\text{epi}}(W)\}.$$

²Relevant domains are explained in all details in Chapter (11).

Furthermore, we prove in Theorem (10.1) on Page (94) and Lemma (14.3) on Page (138) that we can create our intended lineage formulas by mapping relation predicates of the form $R(\bar{c})$ and $R(\bar{c}, _)$ to atomic tuple events on the basis of their corresponding random variables and the truth value F , i.e.,

$$R(\bar{c}) \mapsto X_{R(\bar{c})}(W) \quad \text{and} \quad R(\bar{c}, _) \mapsto F \quad \text{within} \quad \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}.$$

By exploiting these mapping rules and the atomic tuple event labels e_i of Figure (7.3) on Page (51), we can eventually set up our alternative lineage formulas, which are equivalent to the classical ones:

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},1)} &= (R_{\text{epi}}(\text{Dex}, 1, 1) \odot R_{\text{epi}}(\text{Dex}, 1, 2)) \wedge \neg(R_{\text{key}}(\text{Dex}, 1, _) \wedge R_{\text{loc}}(\text{Dex}, _)) \\ &\mapsto \left(\underbrace{X_{R_{\text{epi}}(\text{Dex},1,1)}(W)}_{e_1} \odot \underbrace{X_{R_{\text{epi}}(\text{Dex},1,2)}(W)}_{e_2} \right) \wedge \neg(F \wedge F) \\ &\equiv (e_1 \odot e_2) \wedge \neg F \\ &= \varphi^{(\text{Dex},1)} \\ \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^{(\text{Dex},3)} &= R_{\text{epi}}(\text{Dex}, 3, 3) \wedge \neg(R_{\text{key}}(\text{Dex}, 3, _) \wedge R_{\text{loc}}(\text{Dex}, _)) \\ &\mapsto \underbrace{X_{R_{\text{epi}}(\text{Dex},3,3)}(W)}_{e_3} \wedge \neg(F \wedge F) \\ &\equiv e_3 \wedge \neg F \\ &= \varphi^{(\text{Dex},3)}. \end{aligned}$$

Last but not least, we emphasize again that our described transformation chain has mainly theoretical character.

In practice, we only need to generate relevant domains contained in a single relation and to directly construct our final alternative lineage formulas.

9.3 Vertical lineage construction algorithm

After giving an overview of our theoretical foundations, we now take advantage of them in order to design our vertical lineage construction. It comprises the following:

- the generation of an *event relation* \mathbf{E}_Q within our relational database layer and
- the actual *construction* of all lineage formulas via our second-storage **vlc**-algorithm.

More concretely, we intend to construct our two alternative lineage formulas presented in the previous section:

$$\begin{aligned} \varphi^{(\text{Dex},1)} &= \left(\underbrace{X_{R_{\text{epi}}(\text{Dex},1,1)}(W)}_{e_1} \odot \underbrace{X_{R_{\text{epi}}(\text{Dex},1,2)}(W)}_{e_2} \right) \wedge \neg(F \wedge F) \\ &\equiv (e_1 \odot e_2) \wedge \neg F \\ \varphi^{(\text{Dex},3)} &= \underbrace{X_{R_{\text{epi}}(\text{Dex},3,3)}(W)}_{e_3} \wedge \neg(F \wedge F) \\ &\equiv e_3 \wedge \neg F. \end{aligned}$$

They are derived from their relevant domains:

$$\begin{aligned} \mathbf{D}^{(\text{Dex},1)} &= \{(1, _, _)(x_e, x_k, x_l), (2, _, _)\} \\ \mathbf{D}^{(\text{Dex},3)} &= \{(3, _, _)(x_e, x_k, x_l)\}. \end{aligned}$$

Event relation								
answer tuples			domain values			atomic tuple events		
	t_{x_t}	t_{x_s}	x_e	x_k	x_l	$R_{epi}(t_{x_t}, t_{x_s}, x_e)$	$R_{key}(t_{x_t}, t_{x_s}, x_k)$	$R_{loc}(t_{x_t}, x_l)$
d_1	Dexter	1	1	—	—	e_1	F	F
d_2	Dexter	1	2	—	—	e_2	F	F
d_3	Dexter	3	3	—	—	e_3	F	F

Figure 9.5: First input of **vlc**-algorithm: event relation \mathbf{E}_Q

Relational processing: event relation \mathbf{E}_Q

As depicted in Figure (9.2) on Page (64), our first processing part computes an *event relation* \mathbf{E}_Q via an RDBMS. An event relation \mathbf{E}_Q includes the following three components for our **vlc**-algorithm:

- the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$,
- all relevant domains \mathbf{D}^t , and
- all atomic tuple events e_1, \dots, e_n, F involved in our alternative lineage formulas.

All these ingredients are combined through our **vlc**-algorithm.

To create \mathbf{E}_Q efficiently, we develop in Chapter (13) a set of simple rules, which are only based on relational standard operators. Figure (9.5) on Page (70) depicts the resulting event relation \mathbf{E}_Q built for our example query.

More specifically, in an event relation \mathbf{E}_Q , the columns, which store answer tuples and relevant domain values are named after the variables of the underlying condition structure ϕ^t . In addition, we denote the columns containing atomic tuple events with the relation predicates they are mapped from. Then, they save all atomic tuple events that emerge, if we substitute all variables within the relation predicate of a column by the variable values of a row.

For example, the first row of our event relation in Figure (9.5) on Page (70) carries

$$(\text{Dex}, 1)_{(t_{x_t}, t_{x_s})} \in Q_{\text{poss}(\mathcal{W})} \quad \text{and} \quad (1, _, _)_{(x_e, x_k, x_l)} \in \mathbf{D}^{(\text{Dex}, 1)}.$$

Moreover, it contains e_1, F , and F , since these atomic tuple events are inferred from the relation predicates $R_{epi}(t_{x_t}, t_{x_s}, x_e)$, $R_{key}(t_{x_t}, t_{x_s}, x_k)$ and $R_{loc}(t_{x_t}, x_l)$ as follows:

$$R_{epi}(\text{Dex}, 1, 1) \mapsto \underbrace{X_{R_{epi}(\text{Dex}, 1, 1)}(W)}_{e_1}, \quad R_{key}(\text{Dex}, 1, _) \mapsto F \quad \text{and} \quad R_{loc}(\text{Dex}, _) \mapsto F.$$

Please recall that our theoretical framework proposed following mappings between relation predicates and the random variables used to build atomic tuple events:

$$R(\bar{c}) \mapsto X_{R(\bar{c})} \quad \text{and} \quad R(\bar{c}, _) \mapsto F.$$

Vertical lineage construction algorithm

The classical construction rules of Definition (6.1) on Page (33) propose a recursive combination of subformulas. Given that we represent lineage formulas in the form of formula trees, they create tree structures that grow mainly *horizontally*. They attach smaller formula trees at their root nodes in each construction step in order to get a larger formula tree.

Our construction principle is different. Instead of connecting already built subtrees, we add *complete* tree paths *vertically* to our emerging formula tree.

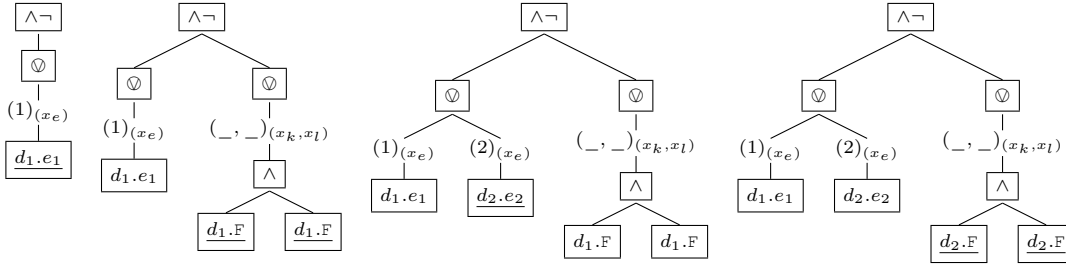


Figure 9.6: Construction of the formula tree for $\varphi^{(\text{Dex},1)}$, where the path-wise inserted atomic tuple events are underlined.

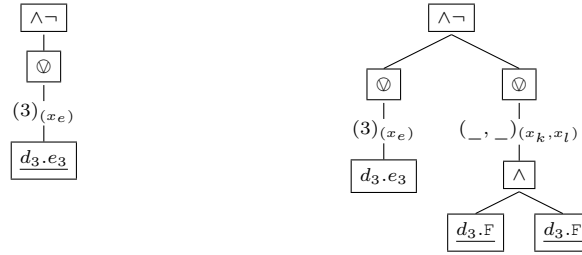


Figure 9.7: Construction of the formula tree for $\varphi^{(\text{Dex},3)}$, where the path-wise inserted atomic tuple events are underlined.

Since each path ends with an atomic tuple event from \mathbf{E}_Q , we can insert all atomic tuple events of \mathbf{E}_Q one-by-one. Therefore, we iterate over all rows of \mathbf{E}_Q and their stored atomic tuple events. The input query Q and its implicit given mapping between algebra operators, logical operators and atomic tuple events help us to find the correct positions for all atomic tuple events of \mathbf{E}_Q within the evolving formula tree.

Figure (9.6) and (9.7) on Page (71) and (71) show the path-wise creation of the two formula trees representing our alternative lineage formulas $\varphi^{(\text{Dex},1)}$ and $\varphi^{(\text{Dex},3)}$. They are built by the two main parts of our construction algorithm shown in Algorithm (1) and (2) on Page (72) and (73), namely:

- main loop function **vlc**(t, \mathbf{E}_Q, Q) iterating over \mathbf{E}_Q and
- recursive function **insertPaths**($d.e, Q, node$) creating paths to our emerging formula tree.

The outcome of **vlc**(t, \mathbf{E}_Q, Q) is an alternative lineage formula φ^t as described in the previous section.

Before we explain both functions in more depth, we take a brief look at the structure of our formula trees to be built. Each of their nodes has one of the following types:

$$node \in \{\boxed{\wedge}, \boxed{\vee}, \boxed{\vee}, \boxed{\wedge \neg}, \boxed{d.e}, \boxed{F}\}.$$

All nodes are connected with pointers, which are heading from the tree root to the tree leaves. More specifically, a binary operator node $\boxed{\wedge}$, $\boxed{\vee}$, or $\boxed{\wedge \neg}$ is always equipped with two pointers referring to its left and right child node labeled as $node.leftChild$ and $node.rightChild$. Furthermore, an n-ary disjunction node $\boxed{\vee}$ manages the n pointers to its child nodes with a map data structure named as $node.children$. The keys of $node.children$ are extracted from the current domain value d , see below.

Main loop in **vlc**(t, \mathbf{E}_Q, Q)

The main loop of **vlc**(t, \mathbf{E}_Q, Q) in Algorithm (1) on Page (72) selects all domain values that belong to the relevant domain \mathbf{D}^t of the considered answer tuple t , see Line (2) and (3). Next, it iterates over all atomic tuple events of the current domain value $d \in \mathbf{D}^t$ in Line (4).

Algorithm 1: $\text{vlc}(t, \mathbf{E}_Q, Q)$

```

1  $rootNode := \boxed{F}$ ;
2 foreach  $d \in \mathbf{E}_Q$  do
3   if  $(d_{\text{vars}(\text{head}(Q))} = t)$  then
4     foreach  $d.e \in \text{atomicTupleEvents}(d)$  do
5        $\text{insertPaths}(d.e, Q, rootNode)$ ;
6     end
7   end
8 end
9 return  $rootNode$ ;

```

For the sake of clarity, we augment an atomic tuple event with its associated domain value to $d.e$. The set of all atomic tuple events for a domain value d is provided through the auxiliary function $\text{atomicTupleEvents}(d)$. For our example, we can obtain following atomic tuple events from our event relation of Figure (9.5) on Page (70):

$$\begin{aligned}
\text{atomicTupleEvents}(d_1) &= \{d_1.e_1, d_1.F\} \\
\text{atomicTupleEvents}(d_2) &= \{d_2.e_2, d_2.F\} \\
\text{atomicTupleEvents}(d_3) &= \{d_3.e_3, d_2.F\}.
\end{aligned}$$

Eventually, our main loop starts in Line (5) the insertion of all paths, which has a leaf $\boxed{d.e}$ involving the current atomic tuple event $d.e$.

Adding Paths via $\text{insertPaths}(d.e, Q, node)$

The main goal of our path creation is to bring the current atomic tuple events $d.e$ to its correct position(s) within our emerging formula tree. For each atomic tuple event $d.e$, our algorithm *recursively* navigates from the tree root to the respective leaves *node-by-node*.

Whenever our algorithm intends to move to a non-existing node, that node has to be created by $\text{createIfAbsent}()$, see Line (5), (9), and (12) in Algorithm (2) on Page (73). The assignment of node types is based on the top-most operator of the current algebra query Q . In Line (3), (7), and (23), we implement our conceptional mapping between algebra operators, logical operators and atomic tuple events presented in Figure (9.3) on Page (65), e.g.,

$$\begin{array}{ccccccc}
\underbrace{Q = \pi_A(Q_1)}_{\text{relational algebra}} & \mapsto & \underbrace{(\bigvee \phi_{Q_1})}_{\text{relevant condition}} & \mapsto & \underbrace{(\bigvee \varphi_{Q_1})}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{\bigvee}}_{\text{tree node}} \\
\underbrace{Q = Q_1 \bowtie Q_2}_{\text{relational algebra}} & \mapsto & \underbrace{(\phi_{Q_1} \wedge \phi_{Q_2})}_{\text{relevant condition}} & \mapsto & \underbrace{(\varphi_{Q_1} \wedge \varphi_{Q_2})}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{\wedge}}_{\text{tree node}} \\
\underbrace{Q = R}_{\text{relational algebra}} & \mapsto & \underbrace{R(\bar{x})}_{\text{relevant condition}} & \mapsto & \underbrace{X_{R(\bar{x})}(W)}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{d.e}}_{\text{tree node}}.
\end{array}$$

Please be aware that we construct tree nodes directly from algebra operators.

The intermediates mappings of our conceptional transformation chain are not materialized in practice.

This also means, we do not have to perform the transformation of our relevant conditions into PNF as discussed before in our theoretical overview.

As mentioned earlier, we navigate through our evolving formula tree node-by-node in order to add a specific path. To be more concrete, this navigation is controlled in Line (5), (9) and (12) by

Algorithm 2: $\text{insertPaths}(d.e, Q, \text{node})$

```

1  switch  $Q$  do
2    case  $Q = \pi_{\mathcal{A}}(Q_1)$ 
3      |  $\text{node} := \boxed{\vee}$ ;
4      |  $\text{key} := d_{\text{vars}(\text{head}(Q_1) \setminus \mathcal{A})}$ ;
5      |  $\text{insertPaths}(d.e, Q_1, \text{createIfAbsent}(\text{node.children}[\text{key}]))$ ;
6    case  $Q = Q_1 \Theta Q_2$  with  $\Theta \in \{\bowtie, \cup, \setminus\}$ 
7      |  $\text{node} := \begin{cases} \boxed{\wedge} & \text{if } \Theta = \bowtie \\ \boxed{\vee} & \text{if } \Theta = \cup \\ \boxed{\wedge \neg} & \text{if } \Theta = \setminus \end{cases}$ 
8      | if  $((\text{relPredicatesMappedFrom}(d.e) \cap \text{relPredicates}(\phi_{Q_1}^t)) \neq \emptyset)$  then
9      | |  $\text{insertPaths}(d.e, Q_1, \text{createIfAbsent}(\text{node.leftChild}))$ ;
10     | end
11     | if  $((\text{relPredicatesMappedFrom}(d.e) \cap \text{relPredicates}(\phi_{Q_2}^t)) \neq \emptyset)$  then
12     | |  $\text{insertPaths}(d.e, Q_2, \text{createIfAbsent}(\text{node.rightChild}))$ ;
13     | end
14    case  $Q = \sigma_F(Q_1)$ 
15      | if  $(F(d) = T)$  then
16      | |  $\text{insertPaths}(d.e, Q_1, \text{node})$ ;
17      | else
18      | |  $\text{node} := \boxed{F}$ ;
19      | end
20    case  $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$ 
21      |  $\text{insertPaths}(d.e, Q_1, \text{node})$ ;
22    case  $Q = R$ 
23      |  $\text{node} := \boxed{d.e}$ ;
24    endsw
25 endsw

```

- the key values of node.children (case $Q = \pi_{\mathcal{A}}(Q_1)$) and
- the two supporting functions **relPredicatesMappedFrom** and **relPredicates** (cases $Q = Q_1 \Theta Q_2$ with $\Theta \in \{\bowtie, \cup, \setminus\}$).

Navigation choice in the case $Q = \pi_{\mathcal{A}}(Q_1)$: The first type of navigation decision has to be chosen, if our algorithm maps a projection operation $\pi_{\mathcal{A}}(Q_1)$ to a n -ary disjunction node $\boxed{\vee}$. In this case, the next node to visit is picked by a key for node.children , which is extracted from the current domain value d , see Line (4).

Our idea for computing this key is to exploit the variables that are projected away by the corresponding projection operation $\pi_{\mathcal{A}}(Q_1)$. The variable values of $\text{vars}(\text{head}(Q_1) \setminus \mathcal{A})$ can serve as proper keys, since they are always unique for a specific \vee -operand.

For instance, in our example query, we have a projection operation

$$\pi_{\mathcal{A}}(Q_1) = \pi_{(\text{tv show}, \text{season})}(\sigma_{(\text{tv show}=\text{Dexter})}(\text{Episodes}))$$

with

$$\begin{aligned}
 \text{vars}(\text{head}(Q_1) \setminus \mathcal{A}) &= \text{vars}(\text{head}(\sigma_{(\text{tv show}=\text{Dexter})}(\text{Episodes})) \setminus \{\text{tv show}, \text{season}\}) \\
 &= \text{vars}(\{\text{tv show}, \text{season}, \text{episode}\} \setminus \{\text{tv show}, \text{season}\}) \\
 &= \text{vars}(\{\text{episode}\}) \\
 &= \{x_e\}.
 \end{aligned}$$

Using the values for x_e , we can uniquely address the corresponding $\boxed{\vee}$ -subtrees of $\varphi^{(\text{Dex}, 1)}$ with $d_{1, x_e} = (1)_{x_e}$ and $d_{2, x_e} = (2)_{x_e}$, see Figure (9.6) on Page (71).

Navigation choice in the cases $Q = Q_1 \Theta Q_2$ with $\Theta \in \{\bowtie, \cup, \setminus\}$: Here, the two auxiliary functions

$$\text{relPredicatesMappedFrom}(d.e) \quad \text{and} \quad \text{relPredicates}(\phi^t)$$

helps our algorithm to decide where to move next.

The first function returns all relation predicates, from which the considered atomic tuple event $d.e$ is conceptionally mapped from. For instance, for the atomic tuple events

$$\begin{aligned} d_1.e_1 &= \{W \in \mathcal{W} \mid X_{R_{epi}(\text{Dex}, 1, 1)}(W)\} \\ d_1.F &= \{W \in \mathcal{W} \mid X_{R_{key}(\text{Dex}, 1, _)}(W)\} = \{W \in \mathcal{W} \mid X_{R_{loc}(\text{Dex}, _)}(W)\}, \end{aligned}$$

we obtain

$$\begin{aligned} \text{relPredicatesMappedFrom}(d_1.e_1) &= \{R_{epi}(t_{x_t}, t_{x_s}, x_e)\} \quad \text{and} \\ \text{relPredicatesMappedFrom}(d_1.F) &= \{R_{key}(t_{x_t}, t_{x_s}, x_k), R_{loc}(t_{x_t}, x_l)\}. \end{aligned}$$

Our second supporter function determines all relation predicates of the current underlying condition structure, e.g.,

$$\begin{aligned} \text{relPredicates}(\phi_Q^t) &= \text{relPredicates}(R_{epi}(t_{x_t}, t_{x_s}, x_k) \wedge \neg(R_{key}(t_{x_t}, t_{x_s}, x_k) \wedge R_{loc}(t_{x_t}, x_l))) \\ &= \{R_{epi}(t_{x_t}, t_{x_s}, x_k), R_{key}(t_{x_t}, t_{x_s}, x_k), R_{loc}(t_{x_t}, x_l)\}. \end{aligned}$$

By intersecting both function outcomes, our algorithm checks in Line (8) and (11) whether the atomic tuple event $d.e$ is mapped from a relation predicate of the current underlying condition structure ϕ_Q^t .

Since a condition structure ϕ_Q^t embodies the logical combination of all relation predicates, it indirectly determines the logical combination of all atomic tuple events. Thus, in each recursive step, we narrow down the respective condition structures based on Q until the correct positions of $d.e$ are reached within our emerging formula tree.

In Theorem (16.1) on Page (169), we prove that our constructed lineage formulas are equivalent to their classical counterparts of Lemma (6.1) on Page (35). To generate the entire set of all lineage formulas, we finally compute

$$\forall t \in \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q) : \varphi_t := \mathbf{vlc}(t, \mathbf{E}_Q, Q).$$

9.4 Summary

To wrap up the first discussion on our concepts, we finally check whether our vertical lineage construction approach is capable to fulfill *all* our initial design goals:

- **Design goal (1): full relational algebra support:** Our query language allows all relational operators, since neither our generation of event relations nor our vertical lineage construction algorithm have any restriction in terms of the given input query.
- **Design goal (2): unlimited lineage formula lengths:** Since our event relations grow vertically in database size, we do not suffer from a length restriction dictated by a relational data type.
Remarkably, we can even generate our event relations in different optimized forms, because our event relations do not need to express any fixed structures of already built lineage formulas.
- **Design goal (3): native relational operators and data types:** All operators and data types which are utilized to generate our event relations can be easily implemented by a standard RDBMS.
- **Design goal (4): relational result set sizes as deterministic case:** In Chapter (13), we verify that the sizes of our event relations do not exceed the bounds known from the deterministic query processing case.

Part IV

Theoretical framework

That part describes the theoretical framework all our developed techniques are relying on. It contains the following chapters:

- Chapter (10) our conceptional main transformation chain and
- Chapter (11); one of our central concepts called *relevant domains*.

Chapter 10

Main transformation chain

In this chapter, we develop the theoretical foundations of our alternative way of constructing lineage formulas.

Please recall that we introduced in Definition (6.1) on Page (33) a lineage formula as a propositional formula representing a complex tuple event of a given probabilistic database. Basically, a lineage formula is built from atomic tuple events connected by logical operators. We already gave examples for lineage formulas in Example (6.2) on Page (36). Please recall that those lineage formulas have been directly built over the structure of an input algebra query.

Concretely, we applied the rules of Lemma (6.1) on Page (35) originally published by Fuhr and Röllecke [39]. In this work, we call them *classical construction rules*.

Moreover, we already motivated in Chapter (9) our central goal to postpone the actual construction of nested lineage formulas into our probabilistic query engine.

In contrast to the classical construction of Lemma (6.1) on Page (35), we propose the alternative construction way illustrated in Figure (10.1) on Page (80). Its main purpose is to facilitate the adjusted basic architecture for our probabilistic database system, see Figure (9.2) on Page (64).

Our conceptional construction approach mainly consists of three transformation steps:

- Main step (1): First, we transform our input algebra query Q into a *domain calculus query* Q^c . The domain calculus query Q^c basically evaluates a *first-order condition* Φ over a set of *domain values* from \mathbf{D} in order to determine its resulting tuples, see Section (10.1).
- Main step (2): Secondly, we rewrite the *first-order condition* Φ of Q^c into a set of equivalent propositional formulas called *relevant conditions* $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. Thereby, a specific relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ is built from its *condition structure* ϕ^t and its *relevant domain* \mathbf{D}^t , see Section (10.2).
- Main step (3): Finally, we map all relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ to lineage formulas φ^t , see Section (10.3).

All theoretical concepts described in this chapter are first and foremost required to prove and verify the ideas behind our more pragmatic algorithms.

As already shown in Chapter (9), we can widely abstract our algorithms from our relative intensive theoretical formalism.

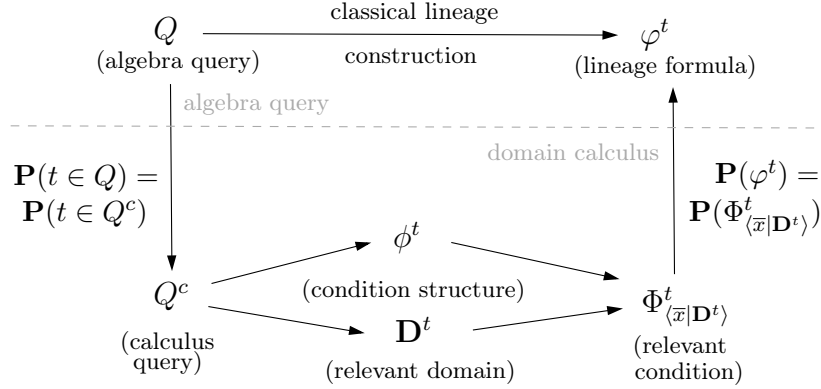


Figure 10.1: Conceptual transformation chain

10.1 Main step (1): Processing an equivalent domain calculus query

The first main pillar of our theoretical framework is the idea of working with a *domain calculus query* instead of an algebra query.

The domain calculus query very naturally supports our goal to divide the overall query processing between RDBMS and probabilistic query engine (Figure (9.2) on Page (64)) by addressing formula structures and domains independently.

Accordingly, we first rewrite our given algebra input query directly into an equivalent domain calculus query. We describe our first transformation by the following topics:

- the basic concepts of the relational domain calculus,
- the derivation of equivalent calculus queries from algebra queries,
- the handling of calculus queries under possible-worlds-semantics.

Relational domain calculus

The *relational domain calculus* is a well-known query language for the relational data model [63]. Next, we recap the basic formalism defining a classical calculus domain query Q^c by discussing the syntax and semantics of Q^c .

Syntax of a domain calculus query

In principle, we follow [74] and clarify the syntactical structure of a domain calculus query.

Definition 10.1 (Syntactic form of a domain calculus query). *We denote a domain calculus \mathcal{DC} as a tuple $\mathcal{DC} = (\mathbf{U}, \mathcal{D}, \bar{x}, \bar{c}, \text{dom}, \mathcal{R}, \Delta)$, where*

- $\mathbf{U} = \{A_1, A_2, \dots\}$ is the universe of all attributes,
- \mathcal{D} is the set of all domains,
- $\bar{x} = \{x_1, x_2, \dots\}$ is a set of variables,
- $\bar{c} = \{c_1, c_2, \dots\}$ is a set of constant names,
- dom is a mapping from $(\mathbf{U} \dot{\cup} \bar{x} \dot{\cup} \bar{c})$ to \mathcal{D} ,

- $\mathcal{R} = \{R_1, R_2, \dots\}$ is a set of relation schemes over \mathbf{U} , and
- $\Delta = \{=, \leq, \dots\}$ is a set of binary-typed operator names.

Then, a domain calculus query Q^c over \mathcal{DC} has the form

$$Q^c := \{\bar{y} \mid \Phi(\bar{y})\},$$

where $\bar{y} \subseteq \bar{x}$. The first-order formula Φ (called condition of Q^c) is built using the following grammar:

$$\Phi ::= x\delta c \mid x_1\delta x_2 \mid c_1\delta c_2 \mid R(\bar{x}, \bar{c}) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid \exists x : \Phi \mid \forall x : \Phi \mid T \mid F,$$

where

- x, x_1 and x_2 represent variables from \bar{x} ,
- c, c_1 and c_2 express constants from \bar{c} ,
- $R(\bar{x}, \bar{c})$ embodies a relational predicate involving variables \bar{x} and constants \bar{c} ,
- δ gives an operator of Δ and
- T/F stands for the truth value true/false.

Additionally, we introduce two convenient notations for domains.

Definition 10.2 (General domain \mathbf{D}). *Let the tuple $\mathcal{DC} = (\mathbf{U}, \mathcal{D}, \bar{x}, \bar{c}, \text{dom}, \mathcal{R}, \Delta)$ be a domain calculus.*

- Then, we denote the domain defined for a variable set $\bar{y} = \{y_1, \dots, y_n\} \subset \bar{x}$ as

$$\mathbf{D}_{\bar{y}} := \text{dom}(y_1) \times \dots \times \text{dom}(y_n).$$

- Moreover, the general domain comprising all variables $\bar{x} = \{x_1, \dots, x_m\}$ of \mathcal{DC} is denoted as

$$\mathbf{D} := \text{dom}(x_1) \times \dots \times \text{dom}(x_m).$$

Remark 10.1 (Abstracting variable/constant names and positions in $R(\bar{x}, \bar{c})$ and $t = (\bar{x}, \bar{c})$). *For the sake of brevity, we often neglect concrete names/positions of variables and constants, when we write a relation predicate as $R(\bar{x}, \bar{c})$ and a tuple as $t = (\bar{x}, \bar{c})$.*

The terms $R(\bar{x}, \bar{c})$ and $t = (\bar{x}, \bar{c})$ mainly indicate that they involve variables and constants. They can be located arbitrarily within the respective argument and value list.

Example 10.1 (Syntactic form of a domain calculus query). *As an example of a domain calculus query, we give Q_{\bowtie}^c , which is equivalent to our example algebra query Q_{\bowtie} of Example (5.1) on Page (30):*

$$Q_{\bowtie}^c = \{x_A \mid \underbrace{\exists x_B : ((R_1(x_A) \wedge R_2(x_A, x_B)) \wedge (R_1(x_A) \wedge R_3(x_A, x_B)))}_{\Phi_{\bowtie}(x_B)}\}.$$

Starting from Definition (10.1) on Page (80), we extend the basic form of Q^c in order to have a dedicated condition Φ^t for each answer tuple t . We finally obtain our alternative lineage formulas φ^t from the set of all conditions Φ^t .

To achieve this goal, we first introduce two new syntactic substitution operations. They can be applied on arbitrary first-order formulas/conditions. Both operations are extensively used in the following chapters of our work.

Definition 10.3 (Syntactic substitution of variables and subconditions). *Let Φ be a first-order formula/condition. Then, we define two syntactic substitution operations denoted as*

$$\Phi_{\langle \bar{x}_1, \bar{x}_2 | \bar{x}_3, \bar{c} \rangle} \quad \text{and} \quad \Phi_{\langle \varphi_1 | \varphi_2 \rangle}.$$

- The first operation $\Phi_{\langle \bar{x}_1, \bar{x}_2 | \bar{x}_3, \bar{c} \rangle}$ replaces all variables of \bar{x}_1 and \bar{x}_2 with variables of \bar{x}_3 and constants of \bar{c} . In the context of syntactic substitutions, we consider $\bar{x}_1, \bar{x}_2, \bar{x}_3$ and \bar{c} as sequences of variables/constants. Accordingly, the concrete replacing between the original variables and substituting variables/constants is determined by their positions within these sequences.
- The second operation $\Phi_{\langle \varphi_1 | \varphi_2 \rangle}$ describes the substitution of a subformula φ_1 through a second subformula φ_2 .

Example 10.2 (Syntactic substitutions). *Let us consider the condition*

$$\Phi = \exists x_A, x_B, x_C : R(x_A, x_B, x_C) \wedge ((x_A = 1) \vee (x_B < 10)) \wedge \neg(x_C = x_A).$$

- Then, $\Phi_{\langle x_A, x_B | 2, 4 \rangle}$ is determined as

$$\Phi_{\langle x_A, x_B | 2, 4 \rangle} = \exists x_C : R(2, 4, x_C) \wedge ((2 = 1) \vee (4 < 10)) \wedge \neg(x_C = 2),$$

where all occurrences of the variables x_A and x_B are replaced with the constants 2 and 4.

- Moreover, the two nested substitutions $(\Phi_{\langle x_A, x_C | x_D, x_D \rangle})_{\langle R(x_D, x_B, x_D) | T \rangle}$ lead to

$$(\Phi_{\langle x_A, x_C | x_D, x_D \rangle})_{\langle R(x_D, x_B, x_D) | T \rangle} = \exists x_B, x_D : T \wedge ((x_D = 1) \vee (x_B < 10)) \wedge \neg(x_D = x_D),$$

since x_A, x_C , and $R(x_D, x_B, x_D)$ are substituted by x_D and T , respectively.

By exploiting our new syntactic substitutions, we are now able to address a single calculus condition Φ^t for each tuple t . To do so, we replace all output variables \bar{y} of Q^c within our general condition Φ by the attribute values of the considered tuple t .

Definition 10.4 (Dedicated condition Φ^t for a tuple t). *Let $Q^c = \{\bar{y} \mid \Phi(\bar{y})\}$ be a domain calculus query and t be a tuple defined over \bar{y} .*

- Then, we define the dedicated calculus condition for t as

$$\Phi^t := \Phi_{\langle \bar{y} | t \rangle}.$$

- Additionally, we rewrite Q^c as

$$Q^c := \{t \mid \Phi^t\}.$$

Example 10.3 (Dedicated condition Φ^t for a tuple t). According to our last definition, we build following two conditions $\Phi_{\bowtie}^{t_1}$ and $\Phi_{\bowtie}^{t_2}$ for the tuples $t_1 = (1)_{x_A}$ and $t_2 = (2)_{x_A}$. They are both inferred from our general condition Φ_{\bowtie} of Example (10.1) on Page (81):

$$\begin{aligned}\Phi_{\bowtie}^{t_1} &= \Phi_{\bowtie}^{(1)} = \Phi_{\bowtie, \langle x_A | 1 \rangle} = \exists x_B : ((R_1(1) \wedge R_2(1, x_B)) \wedge (R_1(1) \wedge R_3(1, x_B))) \\ \Phi_{\bowtie}^{t_2} &= \Phi_{\bowtie}^{(2)} = \Phi_{\bowtie, \langle x_A | 2 \rangle} = \exists x_B : ((R_1(2) \wedge R_2(2, x_B)) \wedge (R_1(2) \wedge R_3(2, x_B))).\end{aligned}$$

Please note that a dedicated condition Φ^t does not include any free variables¹ anymore, since all former output variables are completely replaced by the values of the considered tuple t .

Semantics of a domain calculus query

After defining the syntax of a domain calculus query Q^c , we specify its semantics in three parts, namely:

- the interpretation for all relation schemes, constants, and operations used in the condition Φ^t of Q^c ,
- the determination of the truth value of a condition Φ^t , and
- the final definition of the result set of Q^c .

Definition 10.5. Let $\mathcal{DC} = (\mathbf{U}, \mathcal{D}, \bar{x}, \bar{c}, \text{dom}, \mathcal{R}, \Delta)$ be a domain calculus. Then, an interpretation function I over \mathcal{DC}

- maps any relation schema $R = (A_1, A_2, \dots, A_n)$ from \mathcal{R} to a finite relation:

$$I(R) : \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n),$$

- maps any operation name $\delta \in \Delta$ to a Boolean function:

$$I(\delta) : \text{dom}(\delta) \times \text{dom}(\delta) \rightarrow \{T, F\}, \quad \text{and}$$

- maps any constant name $c \in \bar{c}$ to a domain value:

$$I(c) : \in \text{dom}(c).$$

Next, we compute the truth value of a given condition Φ^t based on a given interpretation I .

Definition 10.6 (Evaluation of a condition Φ^t). Let Φ^t be a condition of domain calculus query from \mathcal{DC} with a corresponding interpretation function I . Then, we recursively evaluate $I^*(\Phi^t) : \in \{T, F\}$ by the following rules:

$$\begin{aligned}\Phi^t = T & : I^*(T) := T \\ \Phi^t = F & : I^*(F) := F \\ \Phi^t = c_1 \delta c_2 & : I^*(c_1 \delta c_2) := I(\delta)(I(c_1), I(c_2)) \\ \Phi^t = R(\bar{c}) & : I^*(R(\bar{c})) := \begin{cases} T & \text{if } I(\bar{c}) \in I(R) \\ F & \text{else} \end{cases} \\ \Phi^t = \Phi_1^t \wedge \Phi_2^t & : I^*(\Phi_1^t \wedge \Phi_2^t) := \begin{cases} T & \text{if } (I^*(\Phi_1^t) = T) \text{ and } (I^*(\Phi_2^t) = T) \\ F & \text{else} \end{cases}\end{aligned}$$

¹A variable is considered as *free*, if it is not bounded by any quantifiers [74].

$$\begin{aligned}
\Phi^t = \Phi_1^t \vee \Phi_2^t & : I^*(\Phi_1^t \vee \Phi_2^t) := \begin{cases} T & \text{if } (I^*(\Phi_1^t) = T) \text{ or } (I^*(\Phi_2^t) = T) \\ F & \text{else} \end{cases} \\
\Phi^t = \neg \Phi_1^t & : I^*(\neg \Phi_1^t) := \begin{cases} T & \text{if } I^*(\Phi_1^t) = F \\ F & \text{else} \end{cases} \\
\Phi^t = \exists y : \Phi_1^t & : I^*(\exists y : \Phi_1^t) := \begin{cases} T & \text{if } d_y \in \mathbf{D}_y \text{ exists with } I^*(\Phi_{1, \langle y | d_y \rangle}^t) = T \\ F & \text{else} \end{cases} \\
\Phi^t = \forall y : \Phi_1^t & : I^*(\forall y : \Phi_1^t) := \begin{cases} T & \text{if } I^*(\neg(\exists y : \neg \Phi_1^t)) = T \\ F & \text{else} \end{cases}.
\end{aligned}$$

Last but not least, we specify the query result of a domain calculus query Q^c .

Definition 10.7 (Semantics of a domain calculus query Q^c). *Let*

$$Q^c = \{\bar{y} \mid \Phi(\bar{y})\} = \{t \mid \Phi^t\}$$

be a domain calculus query over \mathcal{DC} with a corresponding interpretation I . Then, we determine the query result of Q^c as

$$I^*(Q^c) := \{t \in \mathbf{D}_y \mid I^*(\Phi^t) = T\}.$$

Instead of $I^*(Q^c)$ and $I^*(\Phi^t)$, we work in the remainder of this thesis with the equivalence symbol for referring to the query result and the truth value of Q^c and Φ^t .

Definition 10.8 (Equivalences of domain calculus queries and conditions). *Let Q_1 and Q_2 be two algebra queries and*

$$Q_1^c = \{t \mid \Phi_1^t\} \quad \text{and} \quad Q_2^c = \{t \mid \Phi_2^t\}$$

be two domain calculus queries over \mathcal{DC} . If we extend the evaluation function I^ of Definition (10.7) on Page (84) to algebra queries as described in [74], we define*

$$\begin{aligned}
(\Phi_1^t \equiv \Phi_2^t) & :\Leftrightarrow (\forall I : I^*(\Phi_1^t) = I^*(\Phi_2^t)) \\
(Q_1^c \equiv Q_2^c) & :\Leftrightarrow (\forall I : I^*(Q_1^c) = I^*(Q_2^c)) \\
(Q_1 \equiv Q_2^c) & :\Leftrightarrow (\forall I : I^*(Q_1) = I^*(Q_2^c)) \\
(Q_1 \equiv Q_2) & :\Leftrightarrow (\forall I : I^*(Q_1) = I^*(Q_2)).
\end{aligned}$$

Deriving equivalent domain calculus queries from algebra queries

The ability to express each algebra query as an equivalent domain calculus query is a well-known result [74]. To build the bridge between both query languages, we employ a set of recursive mapping rules between algebra and domain calculus operator. From now on, we refer to them as *MAC rules*. By means of the MAC rules, we can always infer an equivalent domain calculus query Q^c from a given algebra query Q .

Lemma 10.1 (Mapping rules between algebra and calculus queries (MAC rules)). *Let Q be an algebra query.*

- Then, an equivalent domain calculus query for Q can be given as

$$Q^c = \{t \mid \Phi_Q^t\},$$

if we define the condition Φ_Q^t by the following recursive mapping rules:

$$\begin{aligned} Q = R & : \Phi_Q^t := R(\bar{z}), \quad \bar{z} \text{ stands for a set of unique variables} \\ Q = \sigma_F(Q_1) & : \Phi_Q^t := \Phi_{Q_1}^t \wedge F \\ Q = \pi_{\mathcal{A}}(Q_1) & : \Phi_Q^t := \exists \bar{z} : \Phi_{Q_1}^t, \quad \bar{z} := \text{vars}(\text{head}(Q_1) \setminus \mathcal{A}) \\ Q = Q_1 \bowtie Q_2 & : \Phi_Q^t := \Phi_{Q_1}^t \wedge (\Phi_{Q_2}^t)_{\langle \text{vars}(\text{head}(Q_1) \cap \text{head}(Q_2)) \mid \text{vars}(\text{head}(Q_1)) \rangle} \\ Q = Q_1 \cup Q_2 & : \Phi_Q^t := \Phi_{Q_1}^t \vee (\Phi_{Q_2}^t)_{\langle \text{vars}(\text{head}(Q_2)) \mid \text{vars}(\text{head}(Q_1)) \rangle} \\ Q = Q_1 \setminus Q_2 & : \Phi_Q^t := \Phi_{Q_1}^t \wedge \neg(\Phi_{Q_2}^t)_{\langle \text{vars}(\text{head}(Q_2)) \mid \text{vars}(\text{head}(Q_1)) \rangle} \\ Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) & : \Phi_Q^t := \Phi_{Q_1}^t_{\langle \text{vars}(\mathcal{A}) \mid \text{vars}(\mathcal{B}) \rangle}. \end{aligned}$$

- In our relation rule $Q = R$, we map all attributes of the relation R to a set of variables \bar{z} . These variables must be unique within the final condition. Accordingly, if a relation R occurs more than once in the overall input query Q , the attributes of the different appearances of R have to be mapped to different variable sets.
- Additionally, we introduce an auxiliary function $\text{vars}(\mathcal{A})$ that returns the variables, which are assigned to the attributes of \mathcal{A} .
- All variables of $\text{head}(Q) \subseteq \bar{x}$ define the output variables of Q^c .

Proof. See [74]. □

Example 10.4 (MAC rules). By applying the MAC rules of Lemma (10.1) on Page (84), we obtain the following equivalent domain calculus queries for our algebra queries Q_{\bowtie} , Q_{\cup} , and Q_{\setminus} of Example (5.1) on Page (30):

$$\begin{aligned} Q_{\bowtie}^c &= \{t \mid \underbrace{\exists x_B : ((R_1(t_{x_A}) \wedge R_2(t_{x_A}, x_B)) \wedge (R_1(t_{x_A}) \wedge R_3(t_{x_A}, x_B)))}_{\Phi_{\bowtie}^t}\} \\ Q_{\cup}^c &= \{t \mid \underbrace{\exists x_B : (R_2(t_{x_A}, x_B) \wedge (x_B = 4)) \vee \exists x'_B : (R_1(t_{x_A}) \wedge R_2(t_{x_A}, x'_B))}_{\Phi_{\cup}^t}\} \\ Q_{\setminus}^c &= \{t \mid \underbrace{\exists x_B : (R_1(t_{x_A}) \wedge R_3(t_{x_A}, x_B)) \wedge \neg(\exists x'_B : R_3(t_{x_A}, x'_B) \wedge (t_{x_A} = 1))}_{\Phi_{\setminus}^t}\}. \end{aligned}$$

As mentioned earlier, the nested structures of our conditions Φ^t are eventually transferred to our alternative lineage formulas φ^t . Since they are mapped from arbitrary relational algebra queries, our approach supports all relational algebra operators.

Remark 10.2 (Relational algebra vs. safe domain calculus). We can transform all algebra queries into equivalent domain calculus queries. The opposite direction is not always possible.

Consequently, there are domain calculus queries which cannot be formulated as equivalent algebra queries, e.g.,

$$Q^c = \{(x_A) \mid \neg R_1(x_A)\}.$$

The subset of domain calculus queries that are equivalent to the set of all relational algebra queries is known as safe domain calculus [74]. Please do not confuse safe domain calculus queries with the safe plans of [24]. These concepts are not related.

Calculus domain queries under possible-worlds-semantics

In Definition (5.2) on Page (30), we presented the semantics of an algebra query evaluated on a probabilistic database. We had to take care of two points in order to determine the answer probability $\mathbf{P}(t \in Q)$ of a given tuple t , namely:

- the evaluation of the given algebra query Q in each world $W \in \mathcal{W}$ (denoted as $Q(W)$) and
- the summing of the probabilities of all worlds W , where the answer tuple t occurs in the query result:

$$\mathbf{P}(t \in Q) := \sum_{W \in \mathcal{W}: t \in Q(W)} \mathbf{P}(W).$$

Next, we apply this principle to domain calculus queries. For this purpose, we first introduce the denotations $Q^c(W)$ and $\Phi^t(W)$ already known from $Q(W)$ to evaluate a domain calculus query and a condition in a specific world $W \in \mathcal{W}$.

Definition 10.9 (Domain calculus query Q^c and condition Φ^t evaluated in a specific world W). *Let $Q^c = \{t \mid \Phi^t\}$ be a domain calculus query with its condition Φ^t over \mathcal{DC} . Then, we say that Q^c and Φ^t are evaluated in the world W (denoted as $Q^c(W)$ and $\Phi^t(W)$), if an interpretation function I of Definition (10.5) on Page (83) maps any relation schema R from \mathcal{R} to the corresponding relation instances $R(W)$ given in world W , i.e.,*

$$I(R) := R(W).$$

Example 10.5 (Domain calculus query Q^c and condition Φ^t evaluated in a specific world W). *To determine the query results and truth values of*

$$Q_{\bowtie}^c(W^{max}), Q_{\cup}^c(W^{max}), Q_{\setminus}^c(W^{max}) \quad \text{and} \quad \Phi_{\bowtie}^{(1)}(W^{max}), \dots, \Phi_{\setminus}^{(2)}(W^{max}),$$

we consider the relation instances from W^{max} of Example (4.1) on Page (24). By doing so, the following truth values can be determined:

$$\begin{aligned} \Phi_{\bowtie}^{(1)}(W^{max}) &= (\exists x_B : (R_1(1) \wedge R_2(1, x_B)) \wedge (R_1(1) \wedge R_3(1, x_B)))(W^{max}) \equiv T \\ \Phi_{\cup}^{(1)}(W^{max}) &= (\exists x_B : (R_2(1, x_B) \wedge (x_B = 4)) \vee \exists x'_B : (R_1(1) \wedge R_2(1, x'_B)))(W^{max}) \equiv T \\ \Phi_{\setminus}^{(1)}(W^{max}) &= (\exists x_B : (R_1(1) \wedge R_3(1, x_B)) \wedge \neg(\exists x'_B : R_3(1, x'_B) \wedge (1 = 1)))(W^{max}) \equiv F \\ \Phi_{\setminus}^{(2)}(W^{max}) &= (\exists x_B : (R_1(2) \wedge R_3(2, x_B)) \wedge \neg(\exists x'_B : R_3(2, x'_B) \wedge (2 = 1)))(W^{max}) \equiv T. \end{aligned}$$

Thus, tuple $(1)_{x_A}$ is contained in the query results of $Q_{\bowtie}^c(W^{max})$ and $Q_{\cup}^c(W^{max})$. Moreover, tuple $(2)_{x_A}$ is part of the query result of $Q_{\setminus}^c(W^{max})$. They all fulfill their respective conditions. On the contrary, the query result of $Q_{\setminus}^c(W^{max})$ does not include tuple $(1)_{x_A}$, since its condition $\Phi_{\setminus}^{(1)}(W^{max})$ is evaluated to F .

Besides specific worlds, we are interested in the likelihood that a calculus condition is satisfied over all worlds. We denote this important probability as $\mathbf{P}(\Phi^t)$. It is the analogon to the answer probability $\mathbf{P}(t \in Q)$ for algebra queries.

Definition 10.10 (Probability of a condition). *Let $Q^c = \{t \mid \Phi^t\}$ be a domain calculus query applied on a probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$.*

- Then, we define the probability of Φ^t as

$$\mathbf{P}(\Phi^t) := \sum_{W \in \mathcal{W}: \Phi^t(W) \equiv \mathbf{T}} \mathbf{P}(W).$$

- In addition, we specify that Φ^t is satisfiable, if

$$\mathbf{P}(\Phi^t) > 0.$$

- Moreover, a condition Φ^t is said to be tractable, if its probability $\mathbf{P}(\Phi^t)$ can be computed in polynomial time considering the formula length of Φ^t .

Our MAC rules laid out in Lemma (10.1) on Page (84) assure that an algebra query Q and its derived calculus query Q^c produce the same result, if we consider both queries in a single world W , i.e., $Q(W) = Q^c(W)$. Unsurprisingly, this property does not change under possible-worlds-semantics.

Lemma 10.2 (Equivalence of algebra and domain calculus queries under possible-worlds-semantics). *Let Q be an algebra query with its equivalent domain calculus query $Q^c = \{t \mid \Phi^t\}$. Then, the results of Q and Q^c yielded on a given probabilistic database \mathbf{pdb} are identical:*

$$\begin{aligned} Q(\mathbf{pdb}) &= (Q_{\text{poss}(\mathcal{W})}, \{pr_t := \mathbf{P}(t \in Q) \mid t \in Q_{\text{poss}(\mathcal{W})}\}) \\ &= (Q_{\text{poss}(\mathcal{W})}^c, \{pr_t := \mathbf{P}(\Phi^t) \mid t \in Q_{\text{poss}(\mathcal{W})}^c\}) = Q^c(\mathbf{pdb}). \end{aligned}$$

Proof. The MAC rules of Lemma (10.1) on Page (84) guarantee that

$$(t \in Q) \Leftrightarrow (\Phi_Q^t \equiv \mathbf{T})$$

for an arbitrary relational database instance. Under possible-worlds-semantics, we then achieve:

$$\forall W \in \mathcal{W} : (t \in Q(W)) \Leftrightarrow (\Phi_Q^t(W) \equiv \mathbf{T}).$$

Consequently, $Q_{\text{poss}(\mathcal{W})}$ equals $Q_{\text{poss}(\mathcal{W})}^c$, and the probability sums

$$\forall t : \mathbf{P}(t \in Q) = \sum_{W \in \mathcal{W}: t \in Q(W)} \mathbf{P}(W) \quad \text{and} \quad \forall t : \mathbf{P}(\Phi^t) = \sum_{W \in \mathcal{W}: \Phi^t(W) \equiv \mathbf{T}} \mathbf{P}(W)$$

are formed over the same set of worlds, i.e.,

$$\forall t : \mathbf{P}(t \in Q) = \mathbf{P}(\Phi^t).$$

□

The last lemma confirms that we can work with domain calculus queries instead of algebra queries. This well-known relationship remains valid under possible-worlds-semantics.

10.2 Main step (2): Setting up relevant conditions over relevant domains

In our second main transformation step, we intend to build relevant conditions over relevant domains. For this task, we introduce another important concept to our framework, namely a *relevant domain* \mathbf{D}^t , which is built for each condition Φ^t .

In detail, a relevant domain contains domain values from the general domain \mathbf{D} , which are sufficient to determine our desired probability $\mathbf{P}(t \in Q)$ via $\mathbf{P}(\Phi^t)$. We determine relevant conditions by the following three steps:

- resolving all variables to make domain values more visible within our conditions,
- identifying common syntactic condition structures and
- defining relevant conditions over relevant domains.

Encoding domain values directly into conditions

In order syntactically incorporate the underlying domain values into a condition Φ^t , we rewrite Φ^t into an equivalent (infinite) propositional counterpart.

For this purpose, we replace all bounded variables in Φ^t with their respective domain values.

The resolving of all quantifiers in Φ^t brings our conditions closer to our target formalism, namely lineage formulas given as *propositional formulas*.

The emerging propositional formula is addressed by the label $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$, which indicates that all variables are substituted with the domain values of \mathbf{D} .

Lemma 10.3 (Propositional condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$). *Let Φ^t be a condition defined over a set of variables $\bar{x} = \{x_1, \dots, x_n\}$ with an overall domain*

$$\mathbf{D} := \mathbf{D}_{x_1} \times \dots \times \mathbf{D}_{x_n}.$$

When we define a propositional condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ by resolving all quantifiers of Φ^t with the recursive rules:

$$\begin{aligned} \Phi^t = R(\bar{c}) &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := R(\bar{c}) \\ \Phi^t = c_1 \delta c_2 &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := c_1 \delta c_2 \\ \Phi^t = T, \Phi^t = F &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := T, \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := F \\ \Phi^t = \Phi_1^t \wedge \Phi_2^t &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := \Phi_{1, \langle \bar{x} | \mathbf{D} \rangle}^t \wedge \Phi_{2, \langle \bar{x} | \mathbf{D} \rangle}^t \\ \Phi^t = \Phi_1^t \vee \Phi_2^t &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := \Phi_{1, \langle \bar{x} | \mathbf{D} \rangle}^t \vee \Phi_{2, \langle \bar{x} | \mathbf{D} \rangle}^t \\ \Phi^t = \neg(\Phi_1^t) &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := \neg(\Phi_{1, \langle \bar{x} | \mathbf{D} \rangle}^t) \\ \Phi^t = \exists y : \Phi_1^t &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := \bigvee_{d_y \in \mathbf{D}_y} (\Phi_{1, \langle \bar{x} | \mathbf{D} \rangle}^t)_{\langle y | d_y \rangle} \\ \Phi^t = \forall y : \Phi_1^t &: \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t := \bigwedge_{d_y \in \mathbf{D}_y} (\Phi_{1, \langle \bar{x} | \mathbf{D} \rangle}^t)_{\langle y | d_y \rangle}, \end{aligned}$$

where \bigvee / \bigwedge represents our n -ary conjunction/disjunction operator of Definition (6.1) on Page (33), then

$$\mathbf{P}(\Phi^t) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t)$$

holds.

Proof. Referring to Definition (10.1) on Page (80), we know that each propositional condition is also a first-order condition, since all propositional conditions constructed by the grammar

$$\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t ::= c_1 \delta c_2 \mid R(\bar{c}) \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \wedge \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \vee \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \mid \neg \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \mid \top \mid \bot$$

can be also generated by the grammar given in Definition (10.1) on Page (80):

$$\Phi ::= x \delta c \mid x_1 \delta x_2 \mid c_1 \delta c_2 \mid R(\bar{x}, \bar{c}) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg \Phi \mid \exists x : \Phi \mid \forall x : \Phi \mid \top \mid \bot.$$

This means that we do not leave our query language.

Except for the quantifiers, we simply transfer the logical operations from the first-order conditions to their propositional counterparts.

Additionally, the quantifier semantics of Definition (10.6) on Page (83) states that a first-order condition $(\exists y : \Phi^t) / (\forall y : \Phi^t)$ holds, if and only if at least one/all substituted subformula(s) $\Phi_{\langle y | d_y \rangle}^t$ is/are fulfilled. This property can be equivalently expressed by n nested \vee / \wedge operations. Then, we simply extend the binary \vee / \wedge -operator to its n -ary version. \square

Because all variables in Φ^t are bounded by quantifiers (see Definition (10.4) on Page (82)), there are no variables left in $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$. As a consequence, $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ embodies a propositional formula only consisting of atomic predicates $R(\bar{c})$ and $(c_1 \delta c_2)$ with $\delta \in \{=, <, >, \dots\}$.

Example 10.6 (Propositional condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$). *If we assume that the domains for our condition $\Phi_{\bowtie}^{(1)}$ of Example (10.3) on Page (83) are given as*

$$\mathbf{D} = \mathbf{D}_{x_B} = \mathbb{N} = \{1, 2, \dots\},$$

we can rewrite our first-order condition $\Phi_{\bowtie}^{(1)}$ into its equivalent propositional form $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)}$:

$$\begin{aligned} \Phi_{\bowtie}^{(1)} &= \exists x_B : (R_1(1) \wedge R_2(1, x_B)) \wedge (R_1(1) \wedge R_3(1, x_B)) \\ &\equiv \bigvee_{d_{x_B} \in \mathbf{D}_{x_B}} ((R_1(1) \wedge R_2(1, d_{x_B})) \wedge (R_1(1) \wedge R_3(1, d_{x_B})))_{\langle x_B | d_{x_B} \rangle} \\ &\equiv ((R_1(1) \wedge R_2(1, 1)) \wedge (R_1(1) \wedge R_3(1, 1))) \vee ((R_1(1) \wedge R_2(1, 2)) \wedge (R_1(1) \wedge R_3(1, 2))) \vee \\ &\quad ((R_1(1) \wedge R_2(1, 3)) \wedge (R_1(1) \wedge R_3(1, 3))) \vee \dots \\ &\equiv \Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)}. \end{aligned}$$

Our last example shows that the formula length of $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ can be infinite, since the cardinality of the underlying domain is usually not limited.

Common syntactic condition structures

Next, we capture common syntactic structures given in all our conditions $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$. The overall syntactic pattern of a condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ is said to be the *condition structure* ϕ^t .

The separation of the overall formula structure contributes to our general goal of constructing lineage formulas independently from domains, see Figure (9.2) on Page (64).

To extract the condition structure ϕ^t from $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$, we isolate the n-ary operators given in $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ by exploiting the transformations known from the *prenex normal form* (PNF) of first-order formulas.

Please remind that a first-order formula is given in PNF, if all quantifiers are located at its beginning. In our case, we demand that all n-ary operators are given in a leading sequence of n-ary operators. It is followed by a n-ary-operator-free subformula.

Definition 10.11 (Condition structure ϕ^t). *Let Q be an algebra query with its equivalent domain calculus query $Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\}$.*

- Then, the condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ is in prenex normal form (PNF), if it is formulated as follows:

$$\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t = \overline{\bigotimes_{d \in \mathbf{D}} \bigotimes} \phi_{\langle \bar{x} | d \rangle}^t.$$

The prefix $\overline{\bigotimes_{d \in \mathbf{D}} \bigotimes}$ stands for a sequence of n-ary operations, and ϕ^t does not involve any n-ary operations.

- The subcondition ϕ^t is called the condition structure of $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$.

The subcondition ϕ^t describes the logical combination of all involved atomic predicates. The syntactic pattern of ϕ^t is thereby identical for all domain values. Hence, we consider ϕ^t as the overall condition structure of $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$.

Example 10.7 (Condition structure ϕ^t). *Our conditions $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)}, \dots, \Phi_{\setminus, \langle \bar{x} | \mathbf{D} \rangle}^{(2)}$ of Example (10.3) on Page (83) have the PNFs:*

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)} &\equiv \overline{\bigotimes_{d \in \mathbf{D}} \bigotimes} \phi_{\bowtie, \langle \bar{x} | d \rangle}^{(1)} \equiv \bigotimes_{(d_{x_B} \in \mathbf{D}_{x_B})} \phi_{\bowtie, \langle x_B | d_{x_B} \rangle}^{(1)} \\ \Phi_{\cup, \langle \bar{x} | \mathbf{D} \rangle}^{(1)} &\equiv \overline{\bigotimes_{d \in \mathbf{D}} \bigotimes} \phi_{\cup, \langle \bar{x} | d \rangle}^{(1)} \equiv \bigotimes_{(d_{x_B} \in \mathbf{D}_{x_B})} \left(\bigotimes_{(d_{x'_B} \in \mathbf{D}_{x'_B})} \phi_{\cup, \langle x'_B | d_{x'_B} \rangle}^{(1)} \right) \langle x_B | d_{x_B} \rangle \\ \Phi_{\setminus, \langle \bar{x} | \mathbf{D} \rangle}^{(1)} &\equiv \overline{\bigotimes_{d \in \mathbf{D}} \bigotimes} \phi_{\setminus, \langle \bar{x} | d \rangle}^{(1)} \equiv \bigotimes_{(d_{x_B} \in \mathbf{D}_{x_B})} \left(\bigotimes_{(d_{x'_B} \in \mathbf{D}_{x'_B})} \phi_{\setminus, \langle x'_B | d_{x'_B} \rangle}^{(1)} \right) \langle x_B | d_{x_B} \rangle \\ \Phi_{\setminus, \langle \bar{x} | \mathbf{D} \rangle}^{(2)} &\equiv \overline{\bigotimes_{d \in \mathbf{D}} \bigotimes} \phi_{\setminus, \langle \bar{x} | d \rangle}^{(2)} \equiv \bigotimes_{(d_{x_B} \in \mathbf{D}_{x_B})} \left(\bigotimes_{(d_{x'_B} \in \mathbf{D}_{x'_B})} \phi_{\setminus, \langle x'_B | d_{x'_B} \rangle}^{(2)} \right) \langle x_B | d_{x_B} \rangle \end{aligned}$$

with the following condition structures:

$$\begin{aligned} \phi_{\bowtie}^{(1)} &= ((R_1(1) \wedge R_2(1, x_B)) \wedge R_1(1) \wedge R_3(1, x_B)) \\ \phi_{\cup}^{(1)} &= (R_2(1, x_B) \wedge (x_B = 4)) \vee (R_1(1) \wedge R_2(1, x'_B)) \\ \phi_{\setminus}^{(1)} &= (R_1(1) \wedge R_3(1, x_B)) \wedge \neg(R_3(1, x'_B) \wedge (1 = 1)) \\ \phi_{\setminus}^{(2)} &= (R_1(2) \wedge R_3(2, x_B)) \wedge \neg(R_3(2, x'_B) \wedge (2 = 1)). \end{aligned}$$

Relevant conditions built over relevant domains

In Example (10.7) on Page (90), we gave propositional conditions $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ in PNF that connect a set of substituted condition structures with n-ary operators. Because the underlying domain \mathbf{D} is infinite in most cases, we apparently achieve infinite conditions as well.

An overarching idea of our approach is to generate domain values within our relational database layer, see Figure (9.2) on Page (64). Therefore, we develop finite domain representations that can be efficiently computed via relational standard operators.

To achieve this goal, we first restrict the general domain \mathbf{D} by dividing it into a set of more specific domains. To be more precise, we set up a single *relevant domain* \mathbf{D}^t for each answer tuple t . This special type of domain just comprises domain values that are classified as relevant for calculating the probabilities of our conditions, i.e.,

$$\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t) \quad \text{with} \quad \mathbf{D}^t \subseteq \mathbf{D}.$$

Please remember that we showed in Definition (10.10) on Page (87) that $\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t)$ is defined by a sum of probabilities that iterates over all worlds:

$$\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t) = \sum_{W \in \mathcal{W} : \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W)} \mathbf{P}(W) = \sum_{W \in \mathcal{W} : \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t(W)} \mathbf{P}(W).$$

By referring to the second sum, we can intuitively suppose that only domain values $d \in \mathbf{D}$ for which at least one world $W \in \mathcal{W}$ exists with $\phi_{\langle \bar{x} | d \rangle}^t(W) \equiv \top$ are needed. In other words, if there is no world $W \in \mathcal{W}$ for a specific domain value d with $\phi_{\langle \bar{x} | d \rangle}^t(W) \equiv \top$, then that domain value d can be neglected.

Definition 10.12 (Relevant domain \mathbf{D}^t and relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t\}.$$

- Then, a domain value $d \in \mathbf{D}$ is said to be *relevant*, if there exists at least one world where ϕ^t substituted by d can be satisfied:

$$(d \in \mathbf{D} \text{ is relevant for } t) :\Leftrightarrow (\exists W \in \mathcal{W} : \phi_{\langle \bar{x} | d \rangle}^t(W)).$$

- The relevant domain $\mathbf{D}^t \subseteq \mathbf{D}$ for a specific tuple t is defined as

$$\mathbf{D}^t := \{d \in \mathbf{D} \mid \exists W \in \mathcal{W} : \phi_{\langle \bar{x} | d \rangle}^t(W)\}.$$

- If a domain value d is not relevant ($d \notin \mathbf{D}^t$), we call it *irrelevant*.
- The calculus condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ built over \mathbf{D}^t is called the *relevant condition* for t .

Example 10.8 (Relevant and irrelevant domain values). *We examine the domain values $(4)_{x_B}$ and $(5)_{x_B}$ with regards to our condition $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)}$ of Example (10.7) on Page (90):*

$$\Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)} \equiv \bigvee_{(d_{x_B} \in \mathbf{D}_{x_B})} \phi_{\bowtie, \langle x_B | d_{x_B} \rangle}^{(1)}$$

with its condition structure

$$\phi_{\bowtie}^{(1)} = ((R_1(1) \wedge R_2(1, x_B)) \wedge R_1(1) \wedge R_3(1, x_B)).$$

It is evaluated on our running example database of Example (4.1) on Page (24).

Following Definition (10.12) on Page (91), we have to investigate the substituted condition structures

$$\begin{aligned}\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)} &= ((R_1(1) \wedge R_2(1, 4)) \wedge (R_1(1) \wedge R_3(1, 4))) \\ \phi_{\bowtie, \langle x_B | 5 \rangle}^{(1)} &= ((R_1(1) \wedge R_2(1, 5)) \wedge (R_1(1) \wedge R_3(1, 5)))\end{aligned}$$

in order to decide whether $(4)_{x_B}$ and $(5)_{x_B}$ are relevant or irrelevant.

More concretely, we attempt to find a world where $\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)}$ and $\phi_{\bowtie, \langle x_B | 5 \rangle}^{(1)}$ can be satisfied. For $\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)}$, we easily identify such a world in the form of \hat{W} , see Figure (4.1) on Page (24). All required tuples exist in \hat{W} :

$$(1) \in R_1(\hat{W}), \quad (1, 4) \in R_2(\hat{W}) \quad \text{and} \quad (1, 4) \in R_3(\hat{W}).$$

This implies that domain value $(4)_{x_B}$ is relevant for $t = (1)_{x_A}$ with $(4)_{x_B} \in \mathbf{D}^{(1)}$.

To the contrary, there is no world where our second substituted condition structure $\phi_{\bowtie, \langle x_B | 5 \rangle}^{(1)}$ can be fulfilled, since the mandatory tuple $(1, 5)$ is not possible in any world. This follows directly from

$$(1, 5) \notin R_2(W^{max}) \quad \text{and} \quad (1, 5) \notin R_3(W^{max}),$$

considering that W^{max} contains the maximal set of possible tuples. Thus, $\phi_{\bowtie, \langle x_B | 5 \rangle}^{(1)}$ cannot be satisfied at all and the domain value $(5)_{x_B}$ is therefore irrelevant.

In Chapter (11), we develop a set of simple rules for generating relevant domains via an RDBMS. By using those rules, we can compute the relevant domain $\mathbf{D}^{(1)}$ for our answer tuple $(1)_{x_A}$ of Q_{\bowtie}^c as

$$\mathbf{D}^{(1)} = \{(3)_{x_B}, (4)\}.$$

It leads to the relevant condition $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)}$:

$$\begin{aligned}\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)} &\equiv \phi_{\bowtie, \langle x_B | 3 \rangle}^{(1)} \vee \phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)} \\ &\equiv ((R_1(1) \wedge R_2(1, 3)) \wedge (R_1(1) \wedge R_3(1, 3))) \vee \\ &\quad ((R_1(1) \wedge R_2(1, 4)) \wedge (R_1(1) \wedge R_3(1, 4))).\end{aligned}$$

In contrast to $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D} \rangle}^{(1)}$ of Example (10.7) on Page (90), our relevant condition $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)}$ not longer relies on the entire domain \mathbf{D} . It is now built over a restricted set of relevant domain values.

Remark 10.3 (Relevant domains vs. active domains). From database theory [74], we know the concepts of an active domain

$$adom(A, R) := \{d \in dom(A) \mid \exists t \in R : t_A = d\}$$

and an extended active domain

$$edom(x, \Phi^t) := \left(\bigcup_{R(x, \bar{y}, \bar{c}) \text{ in } \Phi^t} adom(vars^{-1}(x), R) \right) \cup \{d \in dom(c) \mid (x\delta c) \text{ in } \Phi^t, \delta \in \Delta\}.$$

They both collect domain values that occurring as tuples in the used relations and as constants in an atomic predicate of Φ^t .

For instance, the extended active domain for the variable x_B used in our condition

$$\Phi_{\bowtie}^{(1)} = \exists x_B : ((R_1(1) \wedge R_2(1, x_B)) \wedge (R_1(1) \wedge R_3(1, x_B)))$$

is given by

$$\text{edom}(x_B, \Phi_{\bowtie}^{(1)}) = \text{edom}(B, R_2(W^{max})) \cup \text{edom}(B, R_3(W^{max})) = \{3, 4, 5, 6\}.$$

Since we concluded in Example (10.8) on Page (91) that $(5)_{x_B}$ is irrelevant for $\Phi_{\bowtie}^{(1)}$,

$$\text{edom}(x_B, \Phi_{\bowtie}^{(1)}) \neq \mathbf{D}_{x_B}^{(1)}$$

holds.

Relevant domains do not describe active nor extended active domains.

From this point on, we work on relevant conditions inferred from relevant domains instead of general conditions. Because relevant domains are a central concept of our framework, we study of further important properties of relevant domains in Chapter (11). Here, we focus on explaining our main transformation chain.

10.3 Main step (3): Mapping relevant conditions to lineage formulas

In the previous sections, we described our first two transformation steps. They comprised a transition from an algebra input query Q to an equivalent domain calculus query Q^c , which involves a set of relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$. Next, we present the third part of our alternative construction way. In the last phase of Figure (10.1) on Page (80), we map our relevant conditions to lineage formulas.

In Definition (6.2) on Page (34), we discussed that atomic tuple events within a lineage formula can be modeled by binary random variables. We now make use of these random variables as target for our final mapping between relevant conditions and lineage formulas.

To enable this mapping, we first introduce an adequate set of binary random variables covering all necessary atomic tuple events. Secondly, we apply a simple one-to-one mapping between all atomic predicates of a given relevant condition and our newly created random variables. Lastly, we logically combine all random variables by reusing the syntax of the considered relevant conditions.

Definition 10.13 (Mapping relevant conditions to lineage formulas). *Let $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ be a relevant condition.*

- *Then, we introduce a binary random variable*

$$X_{R(\bar{c})} : \mathcal{W} \rightarrow \{T, F\} \quad / \quad X_{(c_1 \delta c_2)} : \mathcal{W} \rightarrow \{T, F\}$$

for each atomic predicate $R(\bar{c})/(c_1 \delta c_2)$ involved in $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. They are defined as

$$\begin{aligned} \forall W : X_{R(\bar{c})}(W) &:= \begin{cases} T & \text{if } (\bar{c}) \in R(W) \\ F & \text{else} \end{cases} \\ \forall W : X_{(c_1 \delta c_2)}(W) &:= \begin{cases} T & \text{if } (c_1 \delta c_2) \text{ holds in } W \\ F & \text{else.} \end{cases} \end{aligned}$$

- *Respecting the syntactic structure of $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$, we infer a lineage formula φ^t from $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ by mapping all atomic predicates in $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ to their corresponding binary random variables:*

$$R(\bar{c}) \mapsto (X_{R(\bar{c})}(W) = T) \quad \text{and} \quad (c_1 \delta c_2) \mapsto (X_{(c_1 \delta c_2)}(W) = T),$$

which is abbreviated as

$$R(\bar{c}) \mapsto X_{R(\bar{c})}(W) \quad \text{and} \quad (c_1 \delta c_2) \mapsto X_{(c_1 \delta c_2)}(W).$$

Example 10.9 (Mapping relevant conditions to lineage formulas). *Let us consider our relevant condition*

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)} = & ((R_1(1) \wedge R_2(1, 3)) \wedge (R_1(1) \wedge R_3(1, 3))) \oslash \\ & ((R_1(1) \wedge R_2(1, 4)) \wedge (R_1(1) \wedge R_3(1, 4))). \end{aligned}$$

built in Example (10.8) on Page (91).

First, we introduce the binary random variables

$$X_{R_1(1)}, \quad X_{R_2(1,3)}, \quad X_{R_3(1,3)}, \quad X_{R_2(1,4)}, \quad \text{and} \quad X_{R_3(1,4)}$$

corresponding to the atomic predicates in $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)}$. Afterwards, we exploit the syntax of $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)}$ in order set up the lineage formula $\varphi_{\bowtie}^{(1)}$:

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)} \mapsto \varphi_{\bowtie}^{(1)} = & ((X_{R_1(1)}(W) \wedge X_{R_2(1,3)}(W)) \wedge (X_{R_1(1)}(W) \wedge X_{R_3(1,3)}(W))) \oslash \\ & ((X_{R_1(1)}(W) \wedge X_{R_2(1,4)}(W)) \wedge (X_{R_1(1)}(W) \wedge X_{R_3(1,4)}(W))). \end{aligned}$$

In accordance with Definition (6.2) on Page (34), we combine in $\varphi_{\bowtie}^{(1)}$ following identical atomic tuple events

$$\begin{aligned} \{W \in \mathcal{W} \mid X_{R_1(1)}(W)\} &= \{W \in \mathcal{W} \mid (1) \in R_1(W)\} = e_1 \\ \{W \in \mathcal{W} \mid X_{R_2(1,3)}(W)\} &= \{W \in \mathcal{W} \mid (1, 3) \in R_2(W)\} = e_3 \\ \{W \in \mathcal{W} \mid X_{R_3(1,3)}(W)\} &= \{W \in \mathcal{W} \mid (1, 3) \in R_3(W)\} = e_5 \\ \{W \in \mathcal{W} \mid X_{R_2(1,4)}(W)\} &= \{W \in \mathcal{W} \mid (1, 4) \in R_2(W)\} = e_4 \\ \{W \in \mathcal{W} \mid X_{R_3(1,4)}(W)\} &= \{W \in \mathcal{W} \mid (1, 4) \in R_3(W)\} = e_6. \end{aligned}$$

Thereby, the used atomic tuple event labels e_1, \dots, e_5 are taken from Example (6.2) on Page (36). On the basis of e_1, \dots, e_5 , we can eventually reformulate our alternative lineage formula $\varphi_{\bowtie}^{(1)}$ as

$$\begin{aligned} \varphi_{\bowtie}^{(1)} &= ((X_{R_1(1)}(W) \wedge X_{R_2(1,3)}(W)) \wedge (X_{R_1(1)}(W) \wedge X_{R_3(1,3)}(W))) \oslash \\ & \quad ((X_{R_1(1)}(W) \wedge X_{R_2(1,4)}(W)) \wedge (X_{R_1(1)}(W) \wedge X_{R_3(1,4)}(W))) \\ &= ((e_1 \wedge e_3) \wedge (e_1 \wedge e_5)) \oslash ((e_1 \wedge e_4) \wedge (e_1 \wedge e_6)) \\ &\equiv e_1 \wedge ((e_3 \wedge e_5) \oslash (e_4 \wedge e_6)). \end{aligned}$$

This final short form of $\varphi_{\bowtie}^{(1)}$ concludes our alternative construction way.

Theorem 10.1 (Alternative lineage construction). *Let Q be an algebra query evaluated on a probabilistic database $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$. Then, the following three probabilities are identical:*

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \mathbf{P}(\varphi^t),$$

if φ^t is mapped from $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$, see Definition (10.13).

Proof. In Chapter (11), we prove in Lemma (11.2) on Page (99) that relevant conditions are sufficient to compute our desired answer probabilities, i.e.,

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t).$$

Then, it remains to show that $\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$ equals $\mathbf{P}(\varphi^t)$.

To begin with, we know that a relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ under possible-worlds-semantics and its mapped lineage formula φ^t both describe an event of $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$, which has to be identical for proving our proposition, see Definition (6.2) on Page (34) and Definition (10.13) on Page (93).

First, we can be sure that all atomic predicates

$$R(\bar{c})/(c_1 \delta c_2) \quad \text{and their mapped counterparts} \quad X_{R(\bar{c})}(W)/X_{(c_1 \delta c_2)}(W)$$

specify identical atomic tuple events, since they rely on the same evaluation rules, see Definition (10.6) on Page (83) and Definition (10.13) on Page (93).

Secondly, Definition (10.13) on Page (93) guarantees that all atomic predicates of our relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ and alternative lineage formulas φ^t are logically combined by the same underlying syntactic structure. Consequently, they are fulfilled in the same set of worlds:

$$\{W \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W)\} = \{W \mid \varphi^t(W)\}$$

implying

$$\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \sum_{W \in \{W \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W)\}} \mathbf{P}(W) = \sum_{W \in \{W \mid \varphi^t(W)\}} \mathbf{P}(W) = \mathbf{P}(\varphi^t).$$

□

Remark 10.4 (First-order lineage). *To the best of our knowledge, our approach is the first that explicitly exploits the domain calculus for lineage construction. Nonetheless, we want to point out that Dylla, Miliaraki, and Theobald already processed lineage formulas in the form of first-order formulas, which are an essential part of a domain calculus query. They specifically used first-order formulas in conjunction with Datalog techniques for developing an interesting Top-k pruning algorithm based on non-materialized views [32].*

In essence, Dylla et. al abandoned the traditional way of considering lineage formulas purely as propositional formulas as well. However, their ranking algorithm does not focus on lineage construction. They still derive their lineage formulas directly from the structure of the given input query. Concepts similar to our condition structures and relevant domains are not used at all.

10.4 Summary

In this chapter, we laid out the main transformation chain of our alternative lineage construction approach. It consists of three consecutive steps that can be summarized by

- Main step (1): considering the input query as domain calculus query,
- Main step (2): separating condition structures from relevant domains, and
- Main step (3): mapping relevant conditions to lineage formulas.

The main purpose of these steps is to enable our adjusted basic architecture motivated and presented in Chapter (9).

Again, we underline that our three transformation steps have first and foremost a conceptional character. In practice, we only need to compute one relation called *event relation*, which contains all relevant domains, see Chapter (13). The instructions of our vertical construction are also specified over the given input algebra query instead of domain calculus queries. The underlying concepts of our transformation chain mainly assure the correct interplay and outcome of our different query processing parts.

Chapter 11

Relevant domains

In the previous chapter, we introduced relevant domains \mathbf{D}^t as one of the primary concepts of our alternative lineage construction approach. In detail, we utilized them to set up our relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$, which can be exploited in order to compute the desired answer probability of a tuple t :

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t).$$

In the following we turn our attention to the creation of relevant domains. Moreover, a handful of interesting properties of relevant domains are studied. We benefit from these explanations in Part (V), where we eventually implement the query processing of our relational database layer. This chapter contains the following two sections:

- Section (11.1): several important properties of relevant domains and
- Section (11.2): construction rules for a single relevant domain.

11.1 Properties of relevant domains

In this section, we continue our investigation of relevant domains already started in the last chapter. The following topics are examined here in more detail:

- an alternative definition of relevant domains based on so-called *satisfying worlds*,
- the sufficiency of relevant domains for computing answer probabilities,
- the non-minimality of relevant domains, and
- the representation of relevant domain values that range over a whole domain.

Satisfying worlds

For starters, we repeat the main idea of relevant domains by giving an alternative definition. To do so, we introduce a useful tool that also help us in the following chapters of this thesis. It particularly provides the set of worlds, where a specific formula can be satisfied.

Definition 11.1 (Set of satisfying worlds $\text{satWorlds}(\cdot)$). *Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database. If \mathbf{FOL} describes the set of all first-order formulas, we define a function*

$$\text{satWorlds}(\xi) : \mathbf{FOL} \rightarrow 2^{\mathcal{W}}$$

that returns the set of worlds, where the first-order formula ξ is fulfilled:

$$\forall \xi \in \mathbf{FOL} : \text{satWorlds}(\xi) := \{W \in \mathcal{W} \mid \xi(W)\}.$$

Example 11.1 (Satisfying worlds $\text{satWorlds}(\cdot)$). *Let us reconsider our condition structure*

$$\phi_{\bowtie}^{(1)} \equiv ((R_1(1) \wedge R_2(1, x_B)) \wedge R_1(1) \wedge R_3(1, x_B))$$

discussed in Example (10.8) on Page (91). We already showed that

$$\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)}(\hat{W}) \equiv ((R_1(1) \wedge R_2(1, 4)) \wedge (R_1(1) \wedge R_3(1, 4))) (\hat{W}) \equiv \top$$

substituted by the domain value $(4)_{(x_B)}$ is satisfied in world \hat{W} , because

$$(1) \in R_1(\hat{W}), \quad (1, 4) \in R_2(\hat{W}) \quad \text{and} \quad (1, 4) \in R_3(\hat{W}).$$

This means that the world \hat{W} belongs to $\text{satWorlds}(\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)})$.

By using Definition (11.1) on Page (97), we can reformulate our relevance property for domain values.

Definition 11.2 (Alternative definition of relevant domain values). *Let $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ be a condition with its condition structure ϕ^t .*

- *Then, we specify that*

$$(d \in \mathbf{D} \text{ is relevant for } t) :\Leftrightarrow (\text{satWorlds}(\phi_{\langle \bar{x} | d \rangle}^t) \neq \emptyset).$$

- *A relevant domain is then defined as*

$$\mathbf{D}^t := \{d \in \mathbf{D} \mid \text{satWorlds}(\phi_{\langle \bar{x} | d \rangle}^t) \neq \emptyset\}.$$

Example 11.2 (Alternative definition of relevant domain values). *By extending the reasoning of our Examples (11.1), we can clearly see that the domain value $(4)_{(x_B)}$ is relevant, because*

$$\text{satWorlds}(\phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)}) = \{\hat{W}, \dots\} \neq \emptyset.$$

Beyond the alternative definition of relevant domains, it is also possible to reformulate the probability of relevant conditions.

Lemma 11.1 (Probability of a condition). *Let $\Phi^t / \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ be a condition and $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database. Then, we can determine the probability of $\Phi^t / \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ based on \mathbf{pdb} to:*

$$\mathbf{P}(\Phi^t) = \sum_{W \in \text{satWorlds}(\Phi^t)} \mathbf{P}(W) \quad / \quad \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t) = \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t)} \mathbf{P}(W).$$

Proof. The proposition directly follows from Definition (11.1) on Page (97):

$$\text{satWorlds}(\Phi^t) = \{W \in \mathcal{W} \mid \Phi^t(W)\} \quad / \quad \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t) = \{W \in \mathcal{W} \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W)\}.$$

□

Sufficiency of relevant domains

Subsequently, we justify the attribute “relevant” for $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ and \mathbf{D}^t by showing that a relevant domain \mathbf{D}^t is *sufficient* for computing an answer probability $\mathbf{P}(t \in Q)$ via $\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$. Please recall that we already exploited the next lemma in the final proof of our main transformation chain, see Theorem (10.1) on Page (94).

Lemma 11.2 (Relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ determines $P(t \in Q)$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t\}.$$

When $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ is the relevant condition inferred from the relevant domain \mathbf{D}^t , we can compute the answer probability $\mathbf{P}(t \in Q)$ by

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t).$$

Proof. From Lemma (10.2) and (10.3) on Page (87) and (88), we know that $\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t)$. Accordingly, our proposition follows, when the two probability sums

$$\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t) = \sum_{W \in \mathcal{W} : \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W)} \mathbf{P}(W) \quad \text{and} \quad \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \sum_{W \in \mathcal{W} : \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W)} \mathbf{P}(W),$$

are built over the same set of worlds, which is implied by

$$\forall W \in \mathcal{W} : (\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) \Leftrightarrow \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W)).$$

We verify the last equivalence by means of an induction proof over the number of n-ary operations in $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$.

I.) Induction basis ($n = 0$ n-ary operation):

When a propositional calculus condition $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ has no n-ary conjunction/disjunction operations, we know from the MAC rules of Lemma (10.1) on Page (84) that the corresponding first-order calculus condition Φ^t does not involve any variables. Otherwise, an existing variable would be bounded by a quantifier and afterwards resolved by a n-ary operation. Because no variable has to be substituted, the first-order condition and its propositional counterpart are identical:

$$\Phi^t = \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t = \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t,$$

see construction in Lemma (10.3) on Page (88). We can then conclude that our desired equivalence

$$(\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t = \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) \Rightarrow \forall W \in \mathcal{W} : (\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) \Leftrightarrow \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W))$$

holds.

II.) Induction assumption (n n-ary operations):

The equivalence

$$\forall W \in \mathcal{W} : (\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) \Leftrightarrow \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W))$$

is valid for all conditions with n or less n-ary operations.

III.) Induction step ($n + 1$ n-ary operations):

Let $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t}$ be the prenex normal form of $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$. If we split the variable set \bar{x} into $\{x\} \cup \bar{y}$ with $\mathbf{D} = \mathbf{D}_x \times \mathbf{D}_{\bar{y}}$, we can introduce the subcondition

$$\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t := \overline{\bigvee_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t}.$$

Our proposition is already proved for $\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t$ by just considering \bar{y} , see induction assumption. When we also take $\{x\}$ into account, we differentiate two subcases according to the two n-ary operators \bigvee and \bigwedge .

$$\text{Case (1): } \bigvee : \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t} \equiv \bigvee_{d_x \in \mathbf{D}_x} \left(\overline{\bigvee_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t} \right)_{\langle x | d_x \rangle} \equiv \bigvee_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle}$$

By applying our induction assumption (IA), we can directly use our relevant domain $\mathbf{D}_{\bar{y}}^t$ instead of the general domain $\mathbf{D}_{\bar{y}}$, i.e.,

$$\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \bigvee_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \quad \text{instead of} \quad \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \bigvee_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle}.$$

Our key idea (KI) of this subproof is to partition the remaining general domain \mathbf{D}_x into

- the set of relevant domain values \mathbf{D}_x^t taken from \mathbf{D}_x and
- its complementary set of irrelevant domain values $\mathbf{C}_x^t := (\mathbf{D}_x \setminus \mathbf{D}_x^t)$.

According to Definition (10.12) on Page (91), we can be sure that every condition structure, which is substituted by an irrelevant domain value cannot be fulfilled in any world. A disjunctive combination of a set of such substituted condition structures also fails in all worlds. In other words, we can simply ignore these kinds of condition structures and domain values:

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) &\Leftrightarrow \left(\overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t} \right)(W) \\ &\Leftrightarrow \left(\bigvee_{d_x \in \mathbf{D}_x} \left(\overline{\bigvee_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t} \right)_{\langle x | d_x \rangle} \right)(W) \\ &\Leftrightarrow \left(\bigvee_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \right)(W) \\ &\Leftrightarrow \left(\bigvee_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \right)(W) \quad (\text{IA}) \\ &\Leftrightarrow \left(\bigvee_{d_x \in \mathbf{D}_x^t} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \vee \underbrace{\bigvee_{d_x \in \mathbf{C}_x^t} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle}}_{\text{F}} \right)(W) \quad (\text{KI}) \\ &\Leftrightarrow \left(\bigvee_{d_x \in \mathbf{D}_x^t} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \right)(W) \\ &\Leftrightarrow \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W). \end{aligned}$$

$$\text{Case (2): } \bigwedge : \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t} \equiv \bigwedge_{d_x \in \mathbf{D}_x} \left(\overline{\bigvee_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t} \right)_{\langle x | d_x \rangle} \equiv \bigwedge_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle}$$

In this subcase, our key idea (KI) is to exploit the fact that the leading n-ary *conjunction* operation assures

$$\bigwedge_{d_x \in \mathbf{D}_x} \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle} \Leftrightarrow \forall d_x \in \mathbf{D}_x : \left(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t \right)_{\langle x | d_x \rangle}$$

in an arbitrary world. This implies that all domain values are relevant, i.e., $\mathbf{D}_x^t = \mathbf{D}_x$, when there is at least one world $W \in \mathcal{W}$ with $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) \equiv \text{T}$:

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t(W) &\Leftrightarrow \left(\overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t} \right)(W) \\ &\Leftrightarrow \left(\bigwedge_{d_x \in \mathbf{D}_x} \left(\overline{\bigvee_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t} \right)_{\langle x | d_x \rangle} \right)(W) \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \left(\bigoplus_{d_x \in \mathbf{D}_x} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t)_{\langle x | d_x \rangle} \right) (W) \\
&\Leftrightarrow \left(\bigoplus_{d_x \in \mathbf{D}_x} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t)_{\langle x | d_x \rangle} \right) (W) \quad (\text{IA}) \\
&\Leftrightarrow \left(\bigoplus_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}} \rangle}^t)_{\langle x | d_x \rangle} \right) (W) \quad (\text{KI}) \\
&\Leftrightarrow \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t (W)
\end{aligned}$$

□

Lemma (11.2) on Page (99) allows us to concentrate on relevant conditions, if we are determining the answer probabilities $\mathbf{P}(t \in Q)$ via domain calculus queries.

Non-minimality of relevant domains

After showing that relevant domains are sufficient, we can ask whether all of them are necessary. Intuitively, a relevant domain \mathbf{D}^t is minimal, if we cannot remove any domain value d from \mathbf{D}^t without changing the original probability.

Definition 11.3 (Minimal relevant domains). *Let Q be an algebra query with its equivalent domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t\}.$$

Then, a relevant domain \mathbf{D}^t is said to be minimal, if

$$\forall d \in \mathbf{D}^t : \mathbf{P}(\Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \{d\}) \rangle}^t) \neq \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t).$$

holds.

Our next example shows that relevant domains are not minimal in general, i.e., *not* all relevant domain values of \mathbf{D}^t are necessary to compute the answer probability $\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$. Example (11.3) on Page (101) also summarizes the formalism we have introduced so far.

Example 11.3 (Query with non-minimal relevant domain). *To provide a query with a non-minimal relevant domain, let us explore the query*

$$Q \equiv \pi_{\emptyset}(R_1 \bowtie \rho_{(A' \leftarrow A)}(R_1)).$$

Since the attribute list of the outer projection operation of Q does not contain any attributes, we only need to consider the empty tuple $t = ()$ as the only answer tuple.

Our example query is applied on a subdatabase of Example (4.1). This subdatabase only encompasses all possible combinations of tuples from $R_1(W^{max})$ shown in Figure (4.1) on Page (24). It consists of four worlds W^1, W^2, W^3 , and W^4 given by

$$\underbrace{R_1(W^1) = \emptyset}_{W^1}, \quad \underbrace{R_1(W^2) = \{(1)_A\}}_{W^2}, \quad \underbrace{R_1(W^3) = \{(2)_A\}}_{W^3} \quad \text{and} \quad \underbrace{R_1(W^4) = \{(1)_A, (2)_A\}}_{W^4}.$$

In the first step, we set up an equivalent domain calculus query Q^c for Q by

$$Q = \pi_{\emptyset}(R_1 \bowtie \rho_{(A' \leftarrow A)}(R_1)) \equiv \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \Phi^t\} = Q^c,$$

where the first-order and propositional condition Φ^t and $\Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t$ are given by

$$\Phi^t \equiv \exists x_A, x'_A : \phi^t(x_A, x'_A) \quad \text{and} \quad \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t \equiv \bigoplus_{(d_{x_A} \in \mathbf{D}_{x_A})} \left(\bigoplus_{(d_{x'_A} \in \mathbf{D}_{x'_A})} \phi^t_{\langle x'_A | d_{x'_A} \rangle} \right)_{\langle x_A | d_{x_A} \rangle}$$

with their common condition structure

$$\phi^t := R_1(x_A) \wedge R_1(x'_A).$$

Moreover, we assume that the underlying general domain \mathbf{D} is formed by

$$\mathbf{D} = \mathbf{D}_{x_A} \times \mathbf{D}_{x'_A} = \mathbb{N} \times \mathbb{N} = \{(1, 1), (1, 2), \dots\}.$$

On the basis of this setting, our goal is to determine the relevant condition $\Phi_{\langle \mathbf{D}^0 \rangle}^{()}$ for our single answer tuple $t = ()$. Most importantly, we have to clarify its underlying relevant domain \mathbf{D}^0 . To do so, we collect every domain value d for which its substituted condition structure $\phi^t_{\langle x_A, x'_A | d \rangle}$ can be satisfied in at least one world W .

Following Definition (11.1) on Page (97), we can provide the sets of satisfying worlds to our substituted condition structures:

$$\begin{aligned} \text{satWorlds}(\phi_{\langle x_A, x'_A | 1, 1 \rangle}^{()}) &= \{W^2, W^4\}, \\ \text{satWorlds}(\phi_{\langle x_A, x'_A | 1, 2 \rangle}^{()}) &= \{W^4\}, \\ \text{satWorlds}(\phi_{\langle x_A, x'_A | 2, 1 \rangle}^{()}) &= \{W^4\}, \\ \text{satWorlds}(\phi_{\langle x_A, x'_A | 2, 2 \rangle}^{()}) &= \{W^3, W^4\}, \\ \text{satWorlds}(\phi_{\langle x_A, x'_A | d \rangle}^{()}) &= \emptyset \quad \text{for } d_{x_A} \geq 3 \text{ or } d_{x'_A} \geq 3. \end{aligned}$$

By using these substitutions, we simply select all domain values with a non-empty set of satisfying worlds, i.e.,

$$\mathbf{D}^0 = \{d \mid \text{satWorlds}(\phi_{\langle x_A, x'_A | d \rangle}^{()}) \neq \emptyset\} = \{(1, 1)_{(x_A, x'_A)}, (1, 2), (2, 1), (2, 2)\},$$

see also Definition (11.2) on Page (98).

With our relevant domain \mathbf{D}^0 in place, we set up our relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}$:

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()} &\equiv \phi_{\langle x_A, x'_A | 1, 1 \rangle}^{()} \odot \phi_{\langle x_A, x'_A | 1, 2 \rangle}^{()} \odot \phi_{\langle x_A, x'_A | 2, 1 \rangle}^{()} \odot \phi_{\langle x_A, x'_A | 2, 2 \rangle}^{()} \\ &\equiv (R_1(1) \wedge R_1(1)) \odot (R_1(1) \wedge R_1(2)) \odot (R_1(2) \wedge R_1(1)) \odot (R_1(2) \wedge R_1(2)). \end{aligned}$$

Lastly, we determine the probability $\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()})$. For this purpose, we sum up the probabilities of W^2, W^3 , and W^4 , because in these worlds our relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}$ can be satisfied, i.e.,

$$\begin{aligned} \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}) &= \sum_{W \in \mathcal{W}: \Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}(W)} \mathbf{P}(W) = \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()})} \mathbf{P}(W) \\ &= \mathbf{P}(W^2) + \mathbf{P}(W^3) + \mathbf{P}(W^4). \end{aligned}$$

Please remind that we started our current discussion by asking whether all relevant domain values of \mathbf{D}^0 are necessary to compute $\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()})$. The answer to this question is negative, because we can find a smaller relevant domain:

$$\{(1, 1)_{(x_A, x'_A)}, (2, 2)\} \subset \underbrace{\{(1, 1)_{(x_A, x'_A)}, (1, 2), (2, 1), (2, 2)\}}_{\mathbf{D}^0},$$

which makes condition $\Phi_{\langle \bar{x} | \{(1, 1)_{(x_A, x'_A)}, (2, 2)\} \rangle}^{()}$ equivalent to $\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}$:

$$\Phi_{\langle \bar{x} | \{(1, 1)_{(x_A, x'_A)}, (2, 2)\} \rangle}^{()} = \phi_{\langle x_A, x'_A | 1, 1 \rangle}^{()} \odot \phi_{\langle x_A, x'_A | 2, 2 \rangle}^{()}$$

$$\begin{aligned}
&= (R_1(1) \wedge R_1(1)) \oslash (R_1(2) \wedge R_1(2)) \\
&\equiv (R_1(1) \wedge R_1(1)) \oslash (R_1(1) \wedge R_1(2)) \oslash (R_1(2) \wedge R_1(1)) \oslash (R_1(2) \wedge R_1(2)) \\
&= \phi_{\langle x_A, x'_A | 1, 1 \rangle}^{()} \oslash \phi_{\langle x_A, x'_A | 1, 2 \rangle}^{()} \oslash \phi_{\langle x_A, x'_A | 2, 1 \rangle}^{()} \oslash \phi_{\langle x_A, x'_A | 2, 2 \rangle}^{()} \\
&= \Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}.
\end{aligned}$$

The condition $\Phi_{\langle \bar{x} | \{(1,1)_{(x_A, x'_A)}, (2,2)\} \rangle}^{()}$ also covers the worlds W^2 , W^3 and W^4 :

$$\begin{aligned}
\text{satWorlds}(\Phi_{\langle \bar{x} | \{(1,1)_{(x_A, x'_A)}, (2,2)\} \rangle}^{()}) &= \text{satWorlds}(\phi_{\langle x_A, x'_A | 1, 1 \rangle}^{()} \oslash \phi_{\langle x_A, x'_A | 2, 2 \rangle}^{()}) \\
&= \{W^2, W^3, W^4\} \\
&= \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}).
\end{aligned}$$

This implies that the corresponding probabilities are identical

$$\mathbf{P}(\Phi_{\langle \bar{x} | \{(1,1)_{(x_A, x'_A)}, (2,2)\} \rangle}^{()}) = \mathbf{P}(W^2) + \mathbf{P}(W^3) + \mathbf{P}(W^4) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^0 \rangle}^{()}).$$

In essence, we have found a strict subset $\{(1,1)_{(x_A, x'_A)}, (2,2)\} \subset \mathbf{D}^0$, which can be used to compute the answer probability $\mathbf{P}(\cdot \in Q)$, i.e.,

$$\mathbf{P}(\cdot \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \{(1,1)_{(x_A, x'_A)}, (2,2)\} \rangle}^{()}).$$

In addition to Definition (11.3) on Page (101), we present in Section (18.1) a more declarative definition of minimal relevant domains as an advanced topic. It embodies an essential part of the correctness proof of the lineage optimization concept presented later.

Representing relevant domain values that ranging over an entire domain

In the last section, we demonstrated that relevant domains can contain unnecessary domain values.

In contrast to active domains (Remark (10.3) on Page (92)), our relevant domains are not finite in most cases.

Example 11.4 (Query with an infinite relevant domain). *In this example, we explore the running query*

$$Q_{\cup} = \pi_A(\sigma_{(B=4)}(R_2)) \cup \pi_A(R_1 \bowtie R_2) \equiv \{t \mid \Phi_{\cup, \langle \bar{x} | \mathbf{D}^t \rangle}^t\} = Q_{\cup}^c$$

of Example (5.1) on Page (30) in more detail. It exemplifies a typical query with infinite relevant domains.

We are particularly interested in the relevant domain $\mathbf{D}^{(1)}$ for our answer tuple $t = (1)_{x_A}$. In this case, we have to deal with the relevant condition

$$\Phi_{\cup, \langle \bar{x} | \mathbf{D}^t \rangle}^{(1)} \equiv \bigoplus_{(d_{x_B} \in \mathbf{D}_{x_B}^t)} \left(\bigoplus_{(d_{x'_B} \in \mathbf{D}_{x'_B}^t)} \phi_{\cup, \langle x'_B | d_{x'_B} \rangle}^{(1)} \right) \langle x_B | d_{x_B} \rangle$$

with its condition structure

$$\phi_{\cup}^{(1)} \equiv (R_2(1, x_B) \wedge (x_B = 4)) \vee (R_1(1) \wedge R_2(1, x'_B)).$$

By analyzing the structure of $\phi_{\cup}^{(1)}$, we see that $\phi_{\cup}^{(1)}$ combines the two subconditions

$$(R_2(1, x_B) \wedge (x_B = 4)) \quad \text{and} \quad (R_1(1) \wedge R_2(1, x'_B))$$

disjunctively. The semantics of a disjunctive operation assures that the overall truth value $\phi_{\cup}^{(1)} \equiv T$ is already given, if at least one operand is satisfied.

How does this behaviour affect our relevant domains? To answer this question, let us examine the case where the domain value $(4)_{x_B}$ substitutes the variable x_B in $\phi_{\cup}^{(1)}(W^{max})$. That is, the first operand of $\phi_{\cup}^{(1)}(W^{max})$

$$(R_2(1, x_B) \wedge (x_B = 4))_{\langle x_B | 4 \rangle} \equiv (R_2(1, 4) \wedge (4 = 4))$$

is fulfilled, since $(1, 4) \in R_2(W^{max})$. At the same time, our second variable x'_B can range over its whole domain $\mathbf{D}_{x'_B} = \mathbb{N}$ and the overall condition structure $\phi_{\cup, \langle x_B, x'_B | 4, d_{x'_B} \rangle}^{(1)}$ is always satisfied. In other words, the fulfilling of

$$\phi_{\cup, \langle x_B, x'_B | 4, d_{x'_B} \rangle}^{(1)} \equiv (R_2(1, 4) \wedge (4 = 4)) \vee (R_1(1) \wedge R_2(1, d_{x'_B})) \equiv T$$

is independent from the concrete domain value $d_{x'_B}$. For the corresponding relevant subdomain, we then achieve

$$\mathbf{D}_{x'_B}^{(1)} = \mathbf{D}_{x'_B} = \mathbb{N}.$$

We indicate variables with relevant domains which can range over their whole domain by the wildcard symbol $_$.

Definition 11.4 (Representing relevant domain values ranging over an entire domain). *Let $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ be a relevant condition.*

- Then, we represent a relevant domain subset with

$$\{(\bar{c}, \bar{y}) \mid \bar{y} \in \mathbf{D}_{\bar{y}}\} \subseteq \mathbf{D}^t \quad \text{as} \quad (\bar{c}, _) := \{(\bar{c}, \bar{y}) \mid \bar{y} \in \mathbf{D}_{\bar{y}}\}.$$

- Additionally, we use the symbol $_$ within a relation predicate

$$R(\bar{c}, _) := R(\bar{c}, \bar{y}),$$

if it is used in a relevant condition

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \bigotimes_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}^t} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t \quad \text{and} \quad \mathbf{D}_{\bar{y}}^t = \mathbf{D}_{\bar{y}}.$$

Example 11.5 (Representing relevant domain values ranging over an entire domain). *We elaborate our Example (11.4) on Page (103) by expressing the infinite subdomain based on $d_{x_B} = 4$ as*

$$(4, _)_{(x_B, x'_B)} = \{(4, 1)_{(x_B, x'_B)}, (4, 2), (4, 3), \dots\} = \{(4)_{x_B}\} \times \mathbb{N}.$$

The complete relevant domain $\mathbf{D}^{(1)}$ of $\Phi_{\cup, \langle \bar{x} | \mathbf{D}^t \rangle}^{(1)}$ can then be encoded as

$$\mathbf{D}^{(1)} = \{(4, _)_{(x_B, x'_B)}, (_, 3), (_, 4)\}.$$

In the remainder of this work, we use the terms *relevant domains* and *representation of relevant domains* (Definition (11.4) on Page (104)) interchangeably. But we keep in mind that the relevant domain value of the form $d = (\bar{c}, _)$ actually stands for a *set* of relevant domain values.

Last but not least, we would like to point out that Definition (11.4) on Page (104) does not automatically lead to finite relevant domains.

Example 11.6 (Infinite relevant domain). *To provide an example of an infinite relevant domain that exists despite the application of the wildcard symbol $_$, we examine the domain calculus query*

$$Q^c = \{t \mid \exists x_A, x_B : \phi^t(x_A, x_B)\} \equiv \{t \mid \bigvee_{(d_{x_A} \in \mathbf{D}_{x_A})} \left(\bigvee_{(d_{x_B} \in \mathbf{D}_{x_B})} \phi_{\langle x_A, x_B | d \rangle}^t \right)\}$$

with its condition structure

$$\phi^{(1)} = R_1(x_A) \vee \neg R_2(1, x_B) \quad \text{and} \quad \mathbf{D}_{x_A} = \mathbf{D}_{x_B} = \mathbb{N}$$

for the answer tuple $t = (1)$. It is applied on our probabilistic database from Example (4.1) on Page (24).

The corresponding relevant domain is not finite:

$$\mathbf{D}^{(1)} = \{(1, _)(x_A, x_B), (2, _)\} \cup \{(_, 1)(x_A, x_B), (_, 2), (_, 5), (_, 6), \dots\}.$$

Accordingly, we obtain the infinite relevant condition

$$\begin{aligned} \Phi_{\langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)} \equiv & (R_1(1) \vee \neg R_2(1, _)) \otimes (R_1(2) \vee \neg R_2(1, _)) \otimes (R_1(_) \vee \neg R_2(1, 1)) \otimes \\ & (R_1(_) \vee \neg R_2(1, 2)) \otimes (R_1(_) \vee \neg R_2(1, 5)) \otimes (R_1(_) \vee \neg R_2(1, 6)) \otimes \dots \end{aligned}$$

based on $\mathbf{D}^{(1)}$.

As described next, our framework is capable of generating finite relevant domain representations for our query language.

11.2 Generation of relevant domains

In this section, we lay the foundations for determining the relevant domain for a single given tuple t by relational standard operators.

The construction rules devised in this section are further extended in Chapter (13) in order to create an event relation \mathbf{E}_Q containing *all* required relevant domains for a given input query Q .

We next present a basic set of rules for generating a superset $\hat{\mathbf{D}}^t$ of a specific relevant domain \mathbf{D}^t . These rules are derived from the construction of \mathbf{D}^t . The complete development of all construction rules for \mathbf{D}^t and $\hat{\mathbf{D}}^t$ is described in Section (12.1) and (12.2) as an advanced topic.

We generate a superset $\hat{\mathbf{D}}^t$ of \mathbf{D}^t instead of \mathbf{D}^t , since its computation can be efficiently implemented within an RDBMS.

We define our generation rules for $\hat{\mathbf{D}}^t$ directly over the structure of the input query Q .

Lemma 11.3 (Simplified construction rules for superset $\hat{\mathbf{D}}^t$ of \mathbf{D}^t). *Let Q be an algebra query and*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \overline{\bigvee \bigodot} \phi_{\langle \bar{x} \mid d \rangle}^t\}_{d \in \mathbf{D}}$$

be its equivalent domain calculus query. If we generate a set of domain values $\hat{\mathbf{D}}^t$ by using the recursive rules

$$\begin{aligned} Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t &: \hat{\mathbf{D}}_Q^t := \pi_{\bar{x}}(\rho_{(\bar{x} \leftarrow \text{vars}^{-1}(\bar{x}))}(\sigma_{(\text{vars}^{-1}(\bar{y})=t_{\bar{y}})}(R(W^{max})))) \\ Q = \sigma_F(Q_1) &: \hat{\mathbf{D}}_Q^t := \sigma_F(\hat{\mathbf{D}}_{Q_1}^t) \\ Q = \pi_{\mathcal{A}}(Q_1) &: \hat{\mathbf{D}}_Q^t := \hat{\mathbf{D}}_{Q_1}^t \\ Q = Q_1 \bowtie Q_2 &: \hat{\mathbf{D}}_Q^t := \hat{\mathbf{D}}_{Q_1}^{t_1} \bowtie \hat{\mathbf{D}}_{Q_2}^{t_2} \\ Q = Q_1 \cup Q_2 &: \hat{\mathbf{D}}_Q^t := (\hat{\mathbf{D}}_{Q_1}^t \times \{(_)_{\bar{h}_1}\}) \cup (\{(_)_{\bar{h}_2}\} \times \hat{\mathbf{D}}_{Q_2}^t) \\ Q = Q_1 \setminus Q_2 &: \hat{\mathbf{D}}_Q^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ &\quad \mathcal{M}_1 := \hat{\mathbf{D}}_{Q_1}^{t_1} \bowtie \hat{\mathbf{D}}_{Q_2}^{t_2} \\ &\quad \mathcal{M}_2 := (\hat{\mathbf{D}}_{Q_1}^{t_1} \setminus \pi_{\text{attr}(\hat{\mathbf{D}}_{Q_1}^{t_1})}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\} \\ Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) &: \hat{\mathbf{D}}_Q^t := \rho_{(\text{vars}(\mathcal{B}) \leftarrow \text{vars}(\mathcal{A}))}(\hat{\mathbf{D}}_{Q_1}^t), \end{aligned}$$

then

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} \mid \hat{\mathbf{D}}^t \rangle}^t)$$

holds. Thereby,

- *in the relation rule $Q \equiv R$, we refer to the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ from ϕ^t and*
- *in the union and difference rule $Q = Q_1 \cup Q_2$ and $Q = Q_1 \setminus Q_2$, we use our symbol $_$ introduced in Definition (11.4) on Page (104) in order to fill the missing attribute columns:*

$$\bar{h}_1 := (\text{head}(Q_2) \setminus \text{head}(Q_1)) \quad \text{and} \quad \bar{h}_2 := (\text{head}(Q_1) \setminus \text{head}(Q_2)).$$

Proof. See Section (12.1). □

At this point, we discuss in more detail only our relation rule $Q = R$. Its main task is to select all domain values that can satisfy the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ of the underlying condition structure ϕ^t . To provide those domain values, the relation rule conducts four nested operations:

- First, it reads the maximal set of domain values that are suitable for the queried relation as base set. They are given in $R(W^{max})$.
- In the second step, all domain values which are unable to fulfill the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ of $\phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t$ are filtered out given that the answer tuple t is fixed. For this purpose, we only take those column values into account that correspond to our output variables \bar{y} in $R(\bar{x}, t_{\bar{y}})$.
- Subsequently, we rename the original columns with their associated variables \bar{x} of $R(\bar{x}, t_{\bar{y}})$.
- Lastly, our relation rule organizes all selected domain values in columns representing non-output variables of $\Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t$. Please recall that domain values for output variables are directly written into t of $\Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t$. We do not consider them within our relevant domains.

$R_2(W^{max})$			$R_1(W^{max})$		
	A	B		A	
t_3	1	3	t_1	1	
t_4	1	4	t_2	2	

$q_1 = R_2 : \hat{\mathbf{D}}_{q_1}^{(1)} := \pi_{x_B}(\rho_{(x_B \leftarrow B)}(\sigma_{(A=1)}(R_2(W^{max}))))$		
x_B		
3		
4		

$q_2 = R_1 : \hat{\mathbf{D}}_{q_2}^{(1)} := \pi_{\emptyset}(\sigma_{(A=1)}(R_1(W^{max})))$		
()		

$q_3 = \pi_A(\sigma_{(B=4)}(R_2)) : \hat{\mathbf{D}}_{q_3}^{(1)} := \sigma_{(x_B=4)}(\hat{\mathbf{D}}_{q_1}^{(1)})$		
x_B		
4		

$q_4 = \pi_A(R_1 \bowtie R_2) : \hat{\mathbf{D}}_{q_4}^{(1)} := \hat{\mathbf{D}}_{q_2}^{(1)} \bowtie \pi_{x'_B}(\rho_{(x'_B \leftarrow B)}(\sigma_{(A=1)}(R_2(W^{max}))))$		
x'_B		
3		
4		

$Q_{\cup} = \pi_A(\sigma_{B=4}(R_2)) \cup \pi_A(R_1 \bowtie R_2) : \hat{\mathbf{D}}_{\cup}^{(1)} := (\hat{\mathbf{D}}_{q_3}^{(1)} \times \{(_)_{x'_B}\}) \cup (\{(_)_{x_B}\} \times \hat{\mathbf{D}}_{q_4}^{(1)})$		
x_B	x'_B	
4		
—	$\bar{3}$	
—	4	

Figure 11.1: Generation process of the relevant domain $\hat{\mathbf{D}}_{\cup}^{(1)}$ of $\Phi_{\cup, \langle \bar{x} | \hat{\mathbf{D}}_{\cup}^{(1)} \rangle}^{(1)}$

Example 11.7 (Simplified rules for generating $\hat{\mathbf{D}}^t$). When we consider our example query

$$Q_{\cup} = \pi_A(\sigma_{B=4}(R_2)) \cup \pi_A(R_1 \bowtie R_2)$$

with its subqueries

$$q_1 := R_2, \quad q_2 := R_1, \quad q_3 := \pi_A(\sigma_{(B=4)}(R_2)) \quad \text{and} \quad q_4 := \pi_A(R_1 \bowtie R_2)$$

and its underlying condition structure

$$\phi_{\cup}^t = (R_2(t_{x_A}, x_B) \wedge (x_B = 4)) \vee (R_1(t_{x_A}) \wedge R_2(t_{x_A}, x'_B)),$$

we can depict the different steps of generating the relevant domain $\hat{\mathbf{D}}_{\cup}^{(1)}$ for our relevant condition $\Phi_{\langle \bar{x} | \hat{\mathbf{D}}_{\cup}^{(1)} \rangle}^{(1)}$ in Figure (11.1) on Page (107).

In the remainder of this thesis, we do not differentiate between a strict relevant domain \mathbf{D}^t of Definition (10.12) and its superset $\hat{\mathbf{D}}^t$ constructed by Lemma (11.3) on Page (106). We speak in both cases of relevant domains and write \mathbf{D}^t .

$\hat{\mathbf{D}}_{\bowtie}^{(1)}$		$\hat{\mathbf{D}}_{\cup}^{(1)}$			$\hat{\mathbf{D}}_{\setminus}^{(1)}$			$\hat{\mathbf{D}}_{\setminus}^{(2)}$		
	x_B		x_B	x'_B		x_B	x'_B		x_B	x'_B
d_1	3	d_3	4	$\bar{3}$	d_6	3	3	d_{10}	5	—
d_2	4	d_4	—	$\bar{3}$	d_7	3	4	d_{11}	6	—
		d_5	—	4	d_8	4	3			
					d_9	4	4			

Figure 11.2: Supersets of relevant domains for $\Phi_{\bowtie}^{(1)}, \dots, \Phi_{\setminus}^{(2)}$

Example 11.8 (Relevant domains for running example). *Figure (11.2) on Page (108) shows our final (supersets of) relevant domains yielded for our relevant conditions of Example (10.4) on Page (85).*

11.3 Summary

In the last chapter, we studied important properties of relevant domains and gave a set of simple rules for constructing them. In particular, we showed that relevant domains are sufficient, but not minimal with regards to determining answer probabilities. Moreover, we introduced a special notation for relevant domain values that can range over their entire domain.

Chapter 12

Advanced aspects of relevant domains

In this chapter, we discuss the following advanced topics in this chapter:

- Section (12.1): construction rules for an exact relevant domain \mathbf{D}^t and
- Section (12.2): the development of a set of simplified rules generating a superset $\hat{\mathbf{D}}^t$ of \mathbf{D}^t .

12.1 Construction of a relevant domain

Since all our input queries are given as algebra queries, we formulate our construction rules over the structure of Q .

Definition 12.1 (Construction rules for the relevant domain \mathbf{D}^t). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} \mid d \rangle}^t\}.$$

Then, we recursively generate the relevant domain \mathbf{D}^t for $\Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}$ in the following way:

$$\begin{aligned} Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t & : \mathbf{D}_Q^t := \pi_{\bar{x}}(\rho_{(\bar{x} \leftarrow \text{vars}^{-1}(\bar{x}))}(\sigma_{(\text{vars}^{-1}(\bar{y})=t_{\bar{y}})}(R(W^{max})))) \\ Q = \sigma_F(Q_1) & : \mathbf{D}_Q^t := \sigma_F(\mathbf{D}_{Q_1}^t) \\ Q = \pi_A(Q_1) & : \mathbf{D}_Q^t := \mathbf{D}_{Q_1}^t \\ Q = Q_1 \bowtie Q_2 & : \mathbf{D}_Q^t := \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^{t_1} \bowtie \mathbf{D}_{Q_2}^{t_2}) \\ Q = Q_1 \cup Q_2 & : \mathbf{D}_Q^t := (\mathbf{D}_{Q_1}^t \times \{(_)_{\bar{h}_1}\}) \cup (\{(_)_{\bar{h}_2}\} \times \mathbf{D}_{Q_2}^t) \\ Q = Q_1 \setminus Q_2 & : \mathbf{D}_Q^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ & \mathcal{M}_1 := \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^{t_1} \bowtie \mathbf{D}_{Q_2}^{t_2}) \\ & \mathcal{M}_2 := (\mathbf{D}_{Q_1}^{t_1} \setminus \pi_{\text{attr}(\mathbf{D}_{Q_1}^{t_1})}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\} \\ Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) & : \mathbf{D}_Q^t := \rho_{(\text{vars}(\mathcal{B}) \leftarrow \text{vars}(\mathcal{A}))}(\mathbf{D}_{Q_1}^t). \end{aligned}$$

Thereby,

- in the relation rule $Q \equiv R$, we refer to the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ from ϕ^t and
- in the union and difference rule $Q = Q_1 \cup Q_2$ and $Q = Q_1 \setminus Q_2$, we use our symbol $_$ introduced in Definition (11.4) on Page (104) in order to fill the missing attribute columns:

$$\bar{h}_1 := (\text{head}(Q_2) \setminus \text{head}(Q_1)) \quad \text{and} \quad \bar{h}_2 := (\text{head}(Q_1) \setminus \text{head}(Q_2)).$$

In addition to the formal verification of our rules given in Lemma (12.1) on Page (110), we can summarize the basic ideas of our rules as follows:

- **Relation rule** $Q = R$: The main task of our relation rule $Q = R$ is to select all domain values required for the variables occurring in the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$. To provide these domain values, the relation rule conducts four consecutive operations:
 - At first, it takes all domain values that are possible for the queried relation as base set. They are provided in $R(W^{max})$.
 - In the second step, all basic values which are unable to fulfill the corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ of $\phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ are filtered out, if we consider a specific answer tuple t . For that purpose, we only take into account those domain values that correspond to our output variables \bar{y} in $R(\bar{x}, t_{\bar{y}})$.
 - Subsequently, we rename the original columns with their associated variables \bar{x} of $R(\bar{x}, t_{\bar{y}})$.
 - Lastly, our relation rule projects all selected domain values onto columns representing non-output variables of $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. Domain values for output variables are directly written into t of $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. We do not consider them within our relevant domains.
 - **Selection rule** $Q = \sigma_F(Q_1)$: Here, we only exclude all domain values that are not able to fulfill the given selection condition. Such domain values fail in any world. They cannot be relevant.
 - **Projection rule** $Q = \pi_A(Q_1)$: In Lemma (10.1) on Page (84), we specified that the semantics of a projection operation is indirectly mapped to a n-ary disjunction operation via an extensional quantifier. Both operators range over a given (sub)domain. Since our rules determine that specific (sub)domain, our project rule just take over all given domain values.
 - **Join rule** $Q = Q_1 \bowtie Q_2$: In this case, we directly transfer the given join operation to the corresponding operand domains. In addition, a further selection operation (*d is relevant*) is applied on the produced join result. We explain the concrete meaning of this selection in the next section.
 - **Union rule** $Q = Q_1 \cup Q_2$: We know from our MAC rules of Lemma (10.1) on Page (84) that a union operation is expressed on condition level as a logical disjunction. In Section (11.1), we already discussed the associated implications of this kind of operation with regards to the relevant domain values. As a result, we introduced in Definition (11.4) on Page (104) the symbol $_$. It encodes a set of values ranging over their own entire domain. Moreover, we know that the definition of a union operation demands compatible relation schemes from its operands. To meet this constraint, we apply the symbol $_$ in order to fill the values of missing subdomain columns.
 - **Difference rule** $Q = Q_1 \setminus Q_2$: The relevant domain created by the difference rule consists of two parts, namely \mathcal{M}_1 and \mathcal{M}_2 . The first relation \mathcal{M}_1 contains all relevant domain values produced by a join between both operand domains. The second relation \mathcal{M}_2 comprises all those relevant domains of the first operand domain $\mathbf{D}_{Q_1}^t$ that are not already captured by \mathcal{M}_1 .
- Renaming rule** $Q = \rho_{(B \leftarrow A)}(Q_1)$: A renaming operation is simply transferred to the operand domain $\mathbf{D}_{Q_1}^t$ given that the original mapping between attributes and introduced variables is preserved.

Lemma 12.1 (Correctness of the basic rules for constructing \mathbf{D}^t). *Let Q be an algebra query with its equivalent calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \overline{\bigvee_{d \in \mathbf{D}}} \phi_{\langle \bar{x} | d \rangle}^t\}.$$

Then, the rules of Definition (12.1) on Page (109) determine the relevant domain \mathbf{D}^t for a given answer tuple $t \in Q^c$.

Proof. We prove the relevance property of Definition (10.12) on Page (91)

$$d \in \mathbf{D}^t \Leftrightarrow \exists W \in \mathcal{W} : \phi_{\langle \bar{x}|d \rangle}^t(W) \equiv \mathbb{T}$$

by performing an induction over the number n of operators of Q .

I.) Induction basis (Q with $n = 1$ operators, $Q = R$):

In order to build a set of all relevant domain values for a query $Q = R$, we select all tuples from $R(W^{max})$ which have the same attribute values as the considered tuple t . Obviously, the world W^{max} guarantees the relevance of all selected tuples provided by $R(W^{max})$.

II.) Induction assumption (Q with n operators):

The equivalence

$$d \in \mathbf{D}^t \Leftrightarrow \exists W \in \mathcal{W} : \phi_{\langle \bar{x}|d \rangle}^t(W) \equiv \mathbb{T}$$

holds for all algebra queries with n or fewer operators.

III.) Induction step (Q with $n + 1$ operators):

For an algebra query with $(n + 1)$ operators, we have to investigate six cases. They represent the possibilities for the last applied algebra operator in Q .

To begin with, we specify that $\text{vars}(\mathbf{D}^t)$ returns the variables which label the columns of \mathbf{D}^t . If Q_1 and Q_2 are subqueries of Q , then we split the variable set $\bar{x} = \text{vars}(\mathbf{D}_Q^t)$ into two sets

$$\bar{x}_1 := \text{vars}(\mathbf{D}_{Q_1}^t) \quad \text{and} \quad \bar{x}_2 := \text{vars}(\mathbf{D}_{Q_2}^t).$$

The variable sets \bar{x}_1 and \bar{x}_2 are implicitly used in the six cases presented below.

Case (1): $Q = \sigma_F(Q_1)$

Key idea (KI): If $d \in \sigma_F(\mathbf{D}_{Q_1}^t)$, then d has to satisfy the selection condition F . So, $F_{\langle \bar{x}|d \rangle}$ is also fulfilled in the world provided by the induction assumption (IA):

$$\begin{aligned} d \in \sigma_F(\mathbf{D}_{Q_1}^t) &\Leftrightarrow \exists W : (\phi_{Q_1}^t \wedge F)_{\langle \bar{x}|d \rangle}(W) \equiv \mathbb{T} && \text{(IA, KI)} \\ &\Leftrightarrow \exists W : \phi_{Q, \langle \bar{x}|d \rangle}^t(W) \equiv \mathbb{T}. && \text{(MAC)} \end{aligned}$$

Case (2): $Q = \pi_{\mathcal{A}}(Q_1)$:

Key idea (KI): Since our condition structures $\phi_{Q, \langle \bar{x}|d \rangle}^t$ and $\phi_{Q_1, \langle \bar{x}|d \rangle}^t$ are identical, we can directly apply the induction assumption (IA):

$$\begin{aligned} d \in \mathbf{D}_{Q_1}^t &\Leftrightarrow \exists W : \phi_{Q_1, \langle \bar{x}|d \rangle}^t(W) \equiv \mathbb{T} && \text{(IA)} \\ &\Leftrightarrow \exists W : \phi_{Q, \langle \bar{x}|d \rangle}^t(W) \equiv \mathbb{T} && \text{(IA, KI)} \end{aligned}$$

Case (3): $Q = \sigma_{(d \text{ is relevant})}(Q_1 \bowtie Q_2)$

Key idea (KI): If $d \in \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t)$, then the domain value $d = (d_{\bar{x}_1} \bullet d_{\bar{x}_2})$ fulfills the outer selection condition. The operator \bullet is specified in Definition (B.1) on Page (254). Consequently, there is at least one world W where

$$(\phi_{Q_1, \langle \bar{x}_1|d_{\bar{x}_1} \rangle}^t \wedge \phi_{Q_2, \langle \bar{x}_2|d_{\bar{x}_2} \rangle}^t)(W) \equiv \mathbb{T}$$

holds, see our MAC rule for a join operation (Lemma (10.1) on Page (84)). Moreover, we can be sure that there exist at least two worlds W' and W'' where

$$(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t)(W') \equiv \top \quad \text{and} \quad (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W'') \equiv \top$$

are valid, e.g., $W = W' = W''$. This leads to

$$\begin{aligned} d \in \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t) &\Leftrightarrow (d \in (\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t)) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow ((d_{\bar{x}_1} \bullet d_{\bar{x}_2}) \in (\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t)) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow (d_{\bar{x}_1} \in \mathbf{D}_{Q_1}^t) \wedge (d_{\bar{x}_2} \in \mathbf{D}_{Q_2}^t) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow (\exists W : \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \top) \wedge \\ &\quad (\exists W' : \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \top) \wedge \quad \text{(IA)} \\ &\Leftrightarrow (\exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W'') \equiv \top) \quad \text{(KI)} \\ &\Leftrightarrow \exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W'') \equiv \top \quad \text{(KI)} \\ &\Leftrightarrow \exists W'' : \phi_{Q, \langle \bar{x} | d \rangle}^t(W'') \equiv \top. \quad \text{(MAC)} \end{aligned}$$

Case (4): $Q = Q_1 \cup Q_2$:

In accordance with our MAC rule for a union operation (Lemma (10.1) on Page (84)), we can write the corresponding condition structure as

$$\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t.$$

Let us consider a world W where $\phi_{Q, \langle \bar{x} | d \rangle}^t(W)$ is satisfied. We can then differentiate two disjoint subcases derived from the two possible truth values of the first operand.

Case (4.1): $(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W) \equiv \top$ with $\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \top$:

Key idea (KI): The truth value of our first operand $\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \top$ already guarantees the overall truth value $(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W) \equiv \top$. In other words, the overall truth value \top is independent from the second operand $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ and the respective domain values $d_{\bar{x}_2}$ can range over their entire domain $\mathbf{D}_{\bar{x}_2}$, if we presume that \bar{x}_1 and \bar{x}_2 do not overlap. We indicate this property with our symbol $_$ introduced in Definition (11.4) on Page (104):

$$(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W) \equiv (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t)(W).$$

As a result, we only deal with domain values of the form $d = (d_{\bar{x}_1} \bullet (_)_{\bar{x}_2})$ in this subcase. They are generated in the relevant subdomain $(\mathbf{D}_{Q_1}^t \times \{(_)_{\bar{x}_1}\})$ of our union rule from Definition (12.1) on Page (109). When we also take our subcase assumption (SCA) into account, we conclude that

$$\begin{aligned} d \in (\mathbf{D}_{Q_1}^t \times \{(_)_{\bar{x}_1}\}) &\Leftrightarrow (d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}) \in (\mathbf{D}_{Q_1}^t \times \{(_)_{\bar{x}_1}\}) \\ &\Leftrightarrow \exists W : \underbrace{(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t)}_{\top}(W) \equiv \top \quad \text{(IA, KI, SCA)} \\ &\Leftrightarrow \exists W : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t)(W) \equiv \top \quad \text{(KI)} \\ &\Leftrightarrow \exists W : \phi_{Q, \langle \bar{x} | d \rangle}^t(W) \equiv \top \quad \text{(MAC)} \end{aligned}$$

is given.

Case (4.2): $(\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)(W) \equiv \text{T}$ with $\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \text{F}$:

From our subcase assumption (SCA), i.e.,

$$\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \text{F},$$

it follows that the second operand $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ equals T:

$$((\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t) \equiv (\text{F} \vee \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t) \equiv \text{T}) \Rightarrow (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \text{T}).$$

Subsequently, we simply reuse the argumentation of Subcase (4.1) by swapping the roles of $\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t$ and $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$.

Case (5): $Q = Q_1 \setminus Q_2$:

Our MAC rule for a difference operation (Lemma (10.1) on Page (84)) gives us the following condition structure:

$$\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t).$$

The condition structure $\phi_{Q, \langle \bar{x} | d \rangle}^t$ can only be satisfied in a specific world W , if the subformula $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W)$ equals F.

When we assume $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W) \equiv \text{F}$, we can elaborate two disjoint scenarios:

- The first one describes the case where $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W')$ can be fulfilled in at least two different worlds W and W' , i.e., $W \neq W'$.
- Our second subcase presumes that the world W is the only one where $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W) \equiv \text{T}$ is given.

Case (5.1): $(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W) \equiv \text{F}) \wedge (\exists W' \in \mathcal{W} : (W' \neq W) \wedge (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \text{T}))$:

Key idea (KI): By combining the difference rule of Definition (12.1) on Page (109) with the second part of the subcase assumption (SCA), i.e.,

$$\exists W' \in \mathcal{W} : (W' \neq W) \wedge (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \text{T}),$$

we can be sure that $d = (d_{\bar{x}_1} \bullet d_{\bar{x}_2})$ must come from

$$\mathcal{M}_1 := \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t).$$

This also follows from our induction assumption (IA) which assures that $d_{\bar{x}_2}$ is in $\mathbf{D}_{Q_2}^t$, if and only if there is at least one world, where $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ is satisfied. The subcase assumption provides this world in the form of W' .

In addition, the outer selection condition (d is relevant) of \mathcal{M}_1 implies that there is at least one world W'' where the complete condition structure $\phi_Q^t \equiv (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t))$ is fulfilled and we can infer

$$\begin{aligned} d \in \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t) &\Leftrightarrow (d \in (\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t)) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow ((d_{\bar{x}_1} \bullet d_{\bar{x}_2}) \in (\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t)) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow (d_{\bar{x}_1} \in \mathbf{D}_{Q_1}^t) \wedge (d_{\bar{x}_2} \in \mathbf{D}_{Q_2}^t) \wedge (d \text{ is relevant}) \\ &\Leftrightarrow (\exists W : \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \text{T}) \wedge \\ &\quad (\exists W' : \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \text{T}) \wedge \end{aligned}$$

$$\begin{aligned}
& (\exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t))(W'') \equiv \mathbb{T}) \quad (\text{IA, SCA}) \\
& \Leftrightarrow (\exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t))(W'') \equiv \mathbb{T}) \quad (\text{KI}) \\
& \Leftrightarrow \exists W'' : \phi_{Q, \langle \bar{x} | d \rangle}^t(W'') \equiv \mathbb{T}. \quad (\text{MAC})
\end{aligned}$$

Please note that the subcase assumption ensures that the implication

$$\begin{aligned}
& ((\exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t))(W'') \equiv \mathbb{T})) \Rightarrow \\
& ((\exists W : \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \mathbb{T}) \wedge (\exists W' : \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \mathbb{T})) \wedge \\
& (\exists W'' : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t))(W'') \equiv \mathbb{T}))
\end{aligned}$$

is valid. It is needed in order to prove the (\Leftarrow)-direction.

Case (5.2): $(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W) \equiv \mathbb{F}) \wedge (\forall W' : \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W') \equiv \mathbb{F})$:

Key idea (KI): In this subcase the domain value $d_{\bar{x}_2}$ does not fulfill the subcondition $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ in any of the worlds. Thus, we know that $d_{\bar{x}_2}$ cannot come from $\mathbf{D}_{Q_2}^t$. Consequently, it has to be delivered through \mathcal{M}_2 :

$$d \in \mathcal{M}_2 \text{ with } \mathcal{M}_2 := (\mathbf{D}_{Q_1}^t \setminus \pi_{\text{vars}(\mathbf{D}_{Q_1}^t)}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\} \text{ of Definition (12.1).}$$

It follows that

$$\begin{aligned}
d \in \mathcal{M}_2 & \Leftrightarrow (\exists W : \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \mathbb{T}) \wedge (\forall W' : \phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t(W') \equiv \mathbb{F}) \quad (\text{IA, KI}) \\
& \Leftrightarrow \exists W : ((\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W) \equiv \mathbb{T}) \wedge (\phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t(W) \equiv \mathbb{F})) \quad (\text{KI}) \\
& \Leftrightarrow \exists W : (\phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t))(W) \equiv \mathbb{T} \\
& \Leftrightarrow \exists W : \phi_{Q, \langle \bar{x} | d \rangle}^t(W) \equiv \mathbb{T}
\end{aligned}$$

Case (6): $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$:

The MAC rules of Lemma (10.1) on Page (84) determine the condition structure for the renaming operation in the following way:

$$\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv (\phi_{Q_1, \langle \text{vars}(\mathcal{A}) | \text{vars}(\mathcal{B}) \rangle}^t)_{\langle \bar{x} | d \rangle}.$$

Key idea (KI): Here, we make sure that the renaming operation is carry out on all variables as well:

$$\begin{aligned}
d \in \rho_{(\text{vars}(\mathcal{B}) \leftarrow \text{vars}(\mathcal{A}))}(\mathbf{D}_{Q_1}^t) & \Leftrightarrow \exists W : (\phi_{Q_1, \langle \text{vars}(\mathcal{A}) | \text{vars}(\mathcal{B}) \rangle}^t)_{\langle \bar{x} | d \rangle}(W) \equiv \mathbb{T} \quad (\text{IA}) \\
& \Leftrightarrow \exists W : \phi_{Q, \langle \bar{x} | d \rangle}^t(W) \equiv \mathbb{T}. \quad (\text{IA, KI})
\end{aligned}$$

□

12.2 Simplified construction rules

After proving the correctness of our basic rules in the previous section, it is now reasonable to simplify them in order to make them more implementable. First of all, we revisit our join and difference rules introduced in Definition (12.1) on Page (109). They are formulated by means of a selection condition (*d is relevant*) that eliminates irrelevant domain values. We purposely left open a more concrete specification of this condition, because its implementation highly depends

on the class of the considered input query.

Example 12.1 (Selecting relevant domain values within the difference rule (Definition (12.1))).

We examine our example query

$$Q_{\setminus} = \pi_A(R_1 \bowtie R_3) \setminus \pi_A(\sigma_{A=1}(R_3))$$

with its relevant condition

$$\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^t \rangle} = \bigoplus_{(d_{x_B} \in \mathbf{D}_{x_B}^t)} \left(\bigoplus_{(d_{x'_B} \in \mathbf{D}_{x'_B}^t)} \phi_{\setminus, \langle x'_B | d_{x'_B} \rangle}^t \right)_{\langle x_B | d_{x_B} \rangle}$$

and its condition structure

$$\phi_{\setminus}^t = (R_1(t_{x_A}) \wedge R_3(t_{x_A}, x_B)) \wedge \neg(R_3(t_{x_A}, x'_B) \wedge (t_{x_A} = 1)).$$

As usual, we apply Q_{\setminus} on our probabilistic database defined in Example (4.1) on Page (24).

Our primary goal is to set up the relevant domain $\mathbf{D}_{\setminus}^{(1)}$ for the answer tuple $(1)_{(x_A)}$. More concretely, we focus on the difference operation in Q_{\setminus} involving

$$\underbrace{\pi_A(R_1 \bowtie R_3)}_{Q_1} \quad \text{and} \quad \underbrace{\pi_A(\sigma_{A=1}(R_3))}_{Q_2}.$$

It is handled by the difference rule of Definition (12.1) on Page (109):

$$\begin{aligned} Q = Q_1 \setminus Q_2 & : \mathbf{D}_Q^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ \mathcal{M}_1 & := \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_1}^t \bowtie \mathbf{D}_{Q_2}^t) \\ \mathcal{M}_2 & := (\mathbf{D}_{Q_1}^t \setminus \pi_{\text{attr}(\mathbf{D}_{Q_1}^t)}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\}. \end{aligned}$$

In our case, we build

$$\begin{aligned} \mathbf{D}_{Q_1}^{(1)} \bowtie \mathbf{D}_{Q_2}^{(1)} & = \mathbf{D}_{(\pi_A(R_1 \bowtie R_3))}^{(1)} \bowtie \mathbf{D}_{(\pi_A(\sigma_{A=1}(R_3)))}^{(1)} \\ & = \{(3, 3)_{(x_B, x'_B)}, (3, 4), (4, 3), (4, 4)\} \end{aligned}$$

as the base for setting up \mathcal{M}_1 and \mathcal{M}_2 . Since \mathcal{M}_2 is empty, we only consider \mathcal{M}_1 and its outer selection condition (d is relevant).

Please recall that the relevance property of the domain value d checks whether there is at least one world $W \in \mathcal{W}$, where the substituted condition structure $\phi_{\setminus, \langle \bar{x} | d \rangle}^t(W)$ is satisfied, see Definition (10.12) on Page (91) and (11.2) on Page (98):

$$d \in \mathbf{D} \text{ is relevant for } (1)_{(x_A)} : \Leftrightarrow \text{satWorlds}(\phi_{\setminus, \langle \bar{x} | d \rangle}^{(1)}) \neq \emptyset.$$

If we apply that definition on

$$(3, 3)_{(x_B, x'_B)} \quad \text{and} \quad (3, 4)_{(x_B, x'_B)} \quad \text{from} \quad \mathbf{D}_{(\pi_A(R_1 \bowtie R_3))}^t \bowtie \mathbf{D}_{(\pi_A(\sigma_{A=1}(R_3)))}^t,$$

we have to test

$$\text{satWorlds}(\phi_{\setminus, \langle x_B, x_{B'} | 3, 3 \rangle}^{(1)}) \stackrel{?}{\neq} \emptyset \quad \text{and} \quad \text{satWorlds}(\phi_{\setminus, \langle x_B, x_{B'} | 3, 4 \rangle}^{(1)}) \stackrel{?}{\neq} \emptyset.$$

In order to do so, we simplify our substituted conditions under investigation to:

$$\begin{aligned} \phi_{\setminus, \langle x_B, x_{B'} | 3, 3 \rangle}^{(1)} & \equiv (R_1(1) \wedge R_3(1, 3)) \wedge (\neg R_3(1, 3) \vee (1 \neq 1)) \\ & \equiv (R_1(1) \otimes \underbrace{R_3(1, 3) \otimes \neg R_3(1, 3)}_F) \otimes (R_1(1) \otimes R_3(1, 3) \otimes \underbrace{(1 \neq 1)}_F) \end{aligned}$$

$$\equiv F$$

and

$$\begin{aligned}\phi_{\setminus, \langle x_B, x_{B'} \rangle | 3, 4}^{(1)} &\equiv (R_1(1) \wedge R_3(1, 3)) \wedge (\neg R_3(1, 4) \vee (1 \neq 1)) \\ &\equiv (R_1(1) \odot R_3(1, 3) \odot \neg R_3(1, 4)) \odot (R_1(1) \odot R_3(1, 3) \odot \underbrace{(1 \neq 1)}_F) \\ &\equiv R_1(1) \odot R_3(1, 3) \odot \neg R_3(1, 4).\end{aligned}$$

Apparently, the first substituted condition structure $\phi_{\setminus, \langle x_B, x_{B'} \rangle | 3, 3}^{(1)} \equiv F$ is not satisfiable in any world, since the relational predicate $R_3(1, 3)$ occurs in a negated and non-negated form within a n -ary conjunction operation. This means that the domain value $(3, 3)_{(x_B, x_{B'})}$ is irrelevant and needs to be left out.

To the contrary, the second test is positive, because we can give the world

$$\tilde{W} := \{R_1(W^{max}), R_2(W^{max}), R_3(W^{max}) \setminus \{(1, 4)\}, R_4(W^{max})\}$$

based on W^{max} of Figure (4.1) on Page (24) with

$$(1) \in R_1(\tilde{W}), (1, 3) \in R_3(\tilde{W}) \quad \text{and} \quad (1, 4) \notin R_3(\tilde{W})$$

as proof for the relevance of $(3, 4)_{(x_B, x_{B'})}$. Accordingly, the second domain value $(3, 4)_{(x_B, x_{B'})}$ has to be a part of $\mathbf{D}_{\setminus}^{(1)}$.

In general, it can be very costly to conduct a relevance test within an RDBMS for each domain value.

However, there are also combinations of query classes and probabilistic database types, where such tests can be carried out for free. For instance, a **SPJ**-query has no costs, if it is evaluated on a TID database. For this type of queries, all created domain values are always relevant. Any further tests are obsolete.

In cases where checking the selection condition (*d is relevant*) is too costly (see Example (12.1) on Page (115)), we propose to neglect the selection condition (*d is relevant*) completely. Instead, we make use of two simplified rules for our join and difference operator:

$$\begin{aligned}Q = Q_1 \bowtie Q_2 &: \mathbf{D}_Q^t := \mathbf{D}_{Q_1}^{t_1} \bowtie \mathbf{D}_{Q_2}^{t_2} \\ Q = Q_1 \setminus Q_2 &: \mathbf{D}_Q^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ &\mathcal{M}_1 := \mathbf{D}_{Q_1}^{t_1} \bowtie \mathbf{D}_{Q_2}^{t_2} \\ &\mathcal{M}_2 := (\mathbf{D}_{Q_1}^{t_1} \setminus \pi_{\text{attr}(\mathbf{D}_{Q_1}^{t_1})}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\}.\end{aligned}$$

In conjunction with the remaining rules of Definition (12.1) on Page (109), they compute a superset of a relevant domain.

Lemma 12.2 (Simplified construction rules for superset $\hat{\mathbf{D}}^t$ of \mathbf{D}^t). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \rangle | \mathbf{D}}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} \rangle | d}^t\}.$$

If we generate a set of domain values $\hat{\mathbf{D}}^t$ by the recursive rules

$$Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t : \hat{\mathbf{D}}_Q^t := \pi_{\bar{x}}(\rho_{(\bar{x} \leftarrow \text{vars}^{-1}(\bar{x}))}(\sigma_{(\text{vars}^{-1}(\bar{y}) = t_{\bar{y}})}(R(W^{max}))))$$

$$\begin{aligned}
Q = \sigma_F(Q_1) & : \hat{\mathbf{D}}_Q^t := \sigma_F(\hat{\mathbf{D}}_{Q_1}^t) \\
Q = \pi_{\mathcal{A}}(Q_1) & : \hat{\mathbf{D}}_Q^t := \hat{\mathbf{D}}_{Q_1}^t \\
Q = Q_1 \bowtie Q_2 & : \hat{\mathbf{D}}_Q^t := \hat{\mathbf{D}}_{Q_1}^{t_1} \bowtie \hat{\mathbf{D}}_{Q_2}^{t_2} \\
Q = Q_1 \cup Q_2 & : \hat{\mathbf{D}}_Q^t := (\hat{\mathbf{D}}_{Q_1}^t \times \{(_)_{\bar{h}_1}\}) \cup (\{(_)_{\bar{h}_2}\} \times \hat{\mathbf{D}}_{Q_2}^t) \\
Q = Q_1 \setminus Q_2 & : \hat{\mathbf{D}}_Q^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\
& \mathcal{M}_1 := \hat{\mathbf{D}}_{Q_1}^{t_1} \bowtie \hat{\mathbf{D}}_{Q_2}^{t_2} \\
& \mathcal{M}_2 := (\hat{\mathbf{D}}_{Q_1}^{t_1} \setminus \pi_{\text{attr}(\hat{\mathbf{D}}_{Q_1}^{t_1})}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\}, \\
Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) & : \hat{\mathbf{D}}_Q^t := \rho_{(\text{vars}(\mathcal{B}) \leftarrow \text{vars}(\mathcal{A}))}(\hat{\mathbf{D}}_{Q_1}^t),
\end{aligned}$$

where $R(\bar{x}, t_{\bar{y}})$ is taken from ϕ^t and

$$\bar{h}_1 := (\text{head}(Q_2) \setminus \text{head}(Q_1)) \quad \text{and} \quad \bar{h}_2 := (\text{head}(Q_1) \setminus \text{head}(Q_2)),$$

then

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \hat{\mathbf{D}}^t \rangle}^t)$$

holds.

Proof. Let us assume \mathbf{D}_Q^t is constructed by Definition (12.1) on Page (109) and $\hat{\mathbf{D}}_Q^t$ is generated by the rules shown above.

Then, we know from Lemma (11.2) on Page (99) that

$$\mathbf{D}^t \subseteq \hat{\mathbf{D}}^t \quad \text{with} \quad \forall \hat{d} \in (\hat{\mathbf{D}}^t \setminus \mathbf{D}^t) : \hat{d} \text{ is irrelevant}$$

holds, because our simplified rules are obviously less restrictive than our basic rules in Definition (12.1) on Page (109). Please recall that we only removed the selection (d is relevant). This means that unlike in the case of \mathbf{D}^t , we might add irrelevant domain values to $\hat{\mathbf{D}}^t$.

In conjunction with Lemma (10.10) on Page (87) and Lemma (11.2) on Page (99), we can conclude that

$$\begin{aligned}
\mathbf{P}(\Phi_{\langle \bar{x} | \hat{\mathbf{D}}^t \rangle}^t) &= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \hat{\mathbf{D}}^t \rangle}^t)} \mathbf{P}(W) \\
&= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)} \mathbf{P}(W) + \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | (\hat{\mathbf{D}}^t \setminus \mathbf{D}^t) \rangle}^t)} \mathbf{P}(W) \\
&= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)} \mathbf{P}(W) + \sum_{W \in \emptyset} \mathbf{P}(W) \\
&= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)} \mathbf{P}(W) \\
&= \mathbf{P}(t \in Q).
\end{aligned}$$

Thereby, we take into account that a condition substituted by an irrelevant domain value cannot be satisfied in any world, see Definition (10.12) on Page (91). \square

Example 12.2 (Generating $\hat{\mathbf{D}}^t$). Let us consider our example query

$$Q_{\setminus} = \pi_{\mathcal{A}}(R_1 \bowtie R_3) \setminus \pi_{\mathcal{A}}(\sigma_{A=1}(R_3))$$

$R_1(W^{max})$		$R_3(W^{max})$		
	A		A	B
t_1	1	t_5	1	3
t_2	2	t_6	1	4
		t_7	2	5
		t_8	2	6

$q_1 = R_1 : \hat{\mathbf{D}}_{q_1}^{(2)} := \pi_{\emptyset}(\sigma_{(\bar{A}=2)}(R_1(W^{max})))$	
	()

$q_2 = R_3 : \hat{\mathbf{D}}_{q_2}^{(2)} := \pi_{x_B}(\rho_{(x_B \leftarrow B)}(\sigma_{(\bar{A}=2)}(R_3(W^{max}))))$	
	x_B
	5
	6

$q_3 = R_3 : \hat{\mathbf{D}}_{q_3}^{(2)} := \pi_{x'_B}(\rho_{(x'_B \leftarrow B)}(\sigma_{(\bar{A}=2)}(R_3(W^{max}))))$	
	x'_B
	5
	6

$q_4 = \pi_A(R_1 \bowtie R_3) : \hat{\mathbf{D}}_{q_4}^{(2)} := \hat{\mathbf{D}}_{q_1}^{(2)} \bowtie \hat{\mathbf{D}}_{q_2}^{(2)}$	
	x_B
	5
	6

$q_5 = \pi_A(\sigma_{(\bar{A}=1)}(R_3)) : \hat{\mathbf{D}}_{q_5}^{(2)} := \sigma_{(2=1)}(\hat{\mathbf{D}}_{q_3}^{(2)})$	
	x'_B
	\emptyset

$Q_{\setminus} = \pi_A(\sigma_{\bar{A}=1}(R_3)) \setminus \pi_A(R_1 \bowtie R_3) : \hat{\mathbf{D}}_{\setminus}^{(2)} := (\hat{\mathbf{D}}_{q_4}^{(2)} \bowtie \hat{\mathbf{D}}_{q_5}^{(2)}) \cup (\hat{\mathbf{D}}_{q_4}^{(2)} \times (_)_{\bar{h}_4})$	
x_B	x'_B
5	—
6	—

Figure 12.1: Generation process of the relevant domain $\hat{\mathbf{D}}^{(2)}$ of $\Phi_{\setminus, \langle \bar{x} \rangle \hat{\mathbf{D}}^{(2)}}$

with its subqueries

$$q_1 := R_1, \quad q_2 := R_3, \quad q_3 := R_3, \quad q_4 := \pi_A(\sigma_{\bar{A}=1}(R_3)) \quad \text{and} \quad q_5 := \pi_A(R_1 \bowtie R_3)$$

of Example (5.1) on Page (30). Please note that q_2 and q_3 differ from each other in the context of processing Q_{\setminus} , since q_2 and q_3 points to the first and second appearance of R_3 in Q_{\setminus} .

Figure (12.1) on Page (118) demonstrates the construction process of $\hat{\mathbf{D}}_{\setminus}^{(2)}$ using our simplified rules of Lemma (11.3) on Page (106).

Part V

Query processing

After introducing all necessary theoretical foundations in Part (IV), we are now prepared to develop our practical query processing techniques, which can be classified into two groups according to the relational database layer and the probabilistic query engine:

- In Chapter (13), we devise an efficient method for creating event relations as the central component of our relational database layer.
- Subsequently, we develop the following tools for our probabilistic query engine:
 - Chapter (15): vertical lineage construction algorithm,
 - Chapter (17): virtual optimization concept, and
 - Chapter (19): lineage compression technique.

In addition, we present advanced topics for each main technique of this part, see Chapter (14), (16), (18) and (20).

Chapter 13

Event relations

In this chapter, we describe how to efficiently generate event relations. They are created within our relational database layer and serve our vertical construction algorithm as one of the two main inputs.

Please remember that we focused in Chapter (11) on a *single* relevant domain created for a single relevant condition with respect to a certain tuple t . On the basis of Chapter (11), we have to perform several query plans one-by-one in order to generate all required relevant domains.

From a pragmatic view point, we strive for an overall query plan being capable of returning all necessary relevant domains in form of one resulting relation.

We call our aimed relation as *event relation* \mathbf{E}_Q , because it also establishes the connection between relevant domain values and the atomic tuple events used in our final lineage formulas. In essence, three input components are provided by an event relation, namely

- Section (13.2): the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$,
- Section (13.3): all relevant domains \mathbf{D}^t , and
- Section (13.4): all involved atomic tuple events e_1, \dots, e_n, F .

We conclude this chapter with Section (13.5), which outlines our experiments aimed at exploring the relational database layer of several basic approaches.

13.1 Generating event relation \mathbf{E}_Q

In order to specify our event relation \mathbf{E}_Q , we benefit from our groundwork already presented in Part (IV). We basically reuse our simplified rules of Lemma (11.3) on Page (106) and only replace the relation rule

$$Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t : \hat{\mathbf{D}}_Q^t := \pi_{\bar{x}}(\rho_{(\bar{x} \leftarrow \text{vars}^{-1}(\bar{x}))}(\sigma_{(\text{vars}^{-1}(\bar{y})=t_{\bar{y}})}(R(W^{max}))))$$

by

$$Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t : \mathbf{E}_Q := \rho_{((\bar{x}, \bar{y}) \leftarrow \text{vars}^{-1}(\bar{x}, \bar{y}))}(\{(t, e_t) \mid t \in R(W^{max})\}).$$

Our new relation rule initially loads *all* tuples from $R(W^{max})$ into our emerging event relation *without* selecting domain values for a specific answer tuple t . As an important extension, each tuple $t = (\bar{c})$ from $R(W^{max})$ is augmented by its atomic tuple event

$$e_t = \{W \in \mathcal{W} \mid t \in R(W)\}$$

that describes all worlds, where the tuple t exists.

Event relation $\mathbf{E}_{Q_{\bowtie}}$						
	t_{x_A}	x_B	$R_1(t_{x_A})$	$R_2(x_A, x_B)$	$R_3(x_A, x_B)$	
d_1	1	3	$R_1(1) \mapsto e_1$	$R_2(1, 3) \mapsto e_3$	$R_3(1, 3) \mapsto e_5$	
d_2	1	4	$R_1(1) \mapsto e_1$	$R_2(1, 4) \mapsto e_4$	$R_3(1, 4) \mapsto e_6$	

Event relation $\mathbf{E}_{Q_{\cup}}$						
	t_{x_A}	x_B	x'_B	$R_2(x_A, x_B)$	$R_1(x_A)$	$R_2(x_A, x'_B)$
d_3	1	4	—	$R_2(1, 4) \mapsto e_4$	$R_1(1) \mapsto e_1$	$R_2(1, _) \mapsto ?$
d_4	1	—	3	$R_2(1, _) \mapsto ?$	$R_1(1) \mapsto e_1$	$R_2(1, 3) \mapsto e_3$
d_5	1	—	4	$R_2(1, _) \mapsto ?$	$R_1(1) \mapsto e_1$	$R_2(1, 4) \mapsto e_4$

Event relation $\mathbf{E}_{Q_{\setminus}}$						
	t_{x_A}	x_B	x'_B	$R_1(t_{x_A})$	$R_3(t_{x_A}, x_B)$	$R_3(t_{x_A}, x'_B)$
d_6	1	3	3	$R_1(1) \mapsto e_1$	$R_3(1, 3) \mapsto e_5$	$R_3(1, 3) \mapsto e_5$
d_7	1	3	4	$R_1(1) \mapsto e_1$	$R_3(1, 3) \mapsto e_5$	$R_3(1, 4) \mapsto e_6$
d_8	1	4	3	$R_1(1) \mapsto e_1$	$R_3(1, 4) \mapsto e_6$	$R_3(1, 3) \mapsto e_5$
d_9	1	4	4	$R_1(1) \mapsto e_1$	$R_3(1, 4) \mapsto e_6$	$R_3(1, 4) \mapsto e_6$
d_{10}	2	5	—	$R_1(2) \mapsto e_2$	$R_3(2, 5) \mapsto e_7$	$R_3(2, _) \mapsto ?$
d_{11}	2	6	—	$R_1(2) \mapsto e_2$	$R_3(2, 6) \mapsto e_8$	$R_3(2, _) \mapsto ?$

Figure 13.1: Non-final event relations with associated atomic tuple events for Q_{\bowtie} , Q_{\cup} , and Q_{\setminus} of Example (5.1) on Page (30)

We label an additional column carrying atomic tuple events with the corresponding relation predicate $R(\bar{x}, \bar{c})$ of the underlying calculus condition. In Section (13.4), we discuss the purpose of this extension in more detail.

All other rules necessary to construct \mathbf{E}_Q are taken over from our known construction rules of a single relevant domain \mathbf{D}^t given in Lemma (11.3) on Page (106).

Definition 13.1 (Construction rules for an event relation \mathbf{E}_Q). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{(\bar{x}|\mathbf{D})}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{(\bar{x}|d)}^t\}.$$

Then, we generate the event relation \mathbf{E}_Q by

$$\begin{aligned}
Q = R \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t & : \mathbf{E}_Q := \rho_{((\bar{x}, \bar{y}) \leftarrow \text{vars}^{-1}(\bar{x}, \bar{y}))}(\{(t, e_t) \mid t \in R(W^{max})\}) \\
Q = \sigma_F(Q_1) & : \mathbf{E}_Q := \sigma_F(\mathbf{E}_{Q_1}) \\
Q = \pi_{\mathcal{A}}(Q_1) & : \mathbf{E}_Q := \mathbf{E}_{Q_1} \\
Q = Q_1 \bowtie Q_2 & : \mathbf{E}_Q := \mathbf{E}_{Q_1} \bowtie \mathbf{E}_{Q_2} \\
Q = Q_1 \cup Q_2 & : \mathbf{E}_Q := (\mathbf{E}_{Q_1} \times \{(_, F)_{\bar{h}_1}\}) \cup (\{(_, F)_{\bar{h}_2}\} \times \mathbf{E}_{Q_2}) \\
Q = Q_1 \setminus Q_2 & : \mathbf{E}_Q := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\
& \mathcal{M}_1 := \mathbf{E}_{Q_1} \bowtie \mathbf{E}_{Q_2} \\
& \mathcal{M}_2 := (\mathbf{E}_{Q_1} \setminus \pi_{\text{attr}(\mathbf{E}_{Q_1})}(\mathcal{M}_1)) \times \{(_, F)_{\bar{h}_1}\} \\
Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) & : \mathbf{E}_Q := \rho_{(\text{vars}(\mathcal{B}) \leftarrow \text{vars}(\mathcal{A}))}(\mathbf{E}_{Q_1}).
\end{aligned}$$

Thereby,

- in the relation rule $Q = R$, we label the attribute storing e_i with a corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ of ϕ^t and
- in the union and difference rule $Q = Q_1 \cup Q_2$ and $Q = Q_1 \setminus Q_2$, we use our symbol $_$ introduced in Definition (11.4) on Page (104) and the truth value F in order to fill the missing attribute columns:

$$\bar{h}_1 := (\text{attr}(\mathbf{E}_{Q_2}) \setminus \text{attr}(\mathbf{E}_{Q_1})) \quad \text{and} \quad \bar{h}_2 := (\text{attr}(\mathbf{E}_{Q_1}) \setminus \text{attr}(\mathbf{E}_{Q_2})).$$

Example 13.1 (Event relations (not final)). In Figure (13.1) on Page (124), we show the first versions of our event relations $\mathbf{E}_{Q_{\bowtie}}$, $\mathbf{E}_{Q_{\cup}}$, and $\mathbf{E}_{Q_{\setminus}}$. They are generated for our three example queries Q_{\bowtie} , Q_{\cup} and Q_{\setminus} of Example (5.1) on Page (30). They are not given in their final form yet, since we still have to fill the atomic tuple event fields containing a question mark.

13.2 First part of \mathbf{E}_Q : all possible answers $Q_{\text{POSS}(\mathcal{W})}$

First of all, we verify that \mathbf{E}_Q provides the set of all possible answers $Q_{\text{POSS}(\mathcal{W})}$.

Lemma 13.1 (\mathbf{E}_Q includes the set of all possible answers $Q_{\text{POSS}(\mathcal{W})}$). Let Q be an algebra query and let \mathbf{E}_Q be its event relation. Then, the event relation \mathbf{E}_Q includes the set of all possible answers $Q_{\text{POSS}(\mathcal{W})}$:

$$Q_{\text{POSS}(\mathcal{W})} \subseteq \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q).$$

Proof. See Section (14.1). □

The last lemma states that $Q_{\text{POSS}(\mathcal{W})}$ is given in \mathbf{E}_Q in the form of a superset.

We have to filter the candidates contained in \mathbf{E}_Q in order to determine the exact set of all possible answers.

For this task, we take advantage of the probabilities of our final lineage formulas $\mathbf{P}(\varphi^t)$.

Lemma 13.2 (Extracting $Q_{\text{POSS}(\mathcal{W})}$ from \mathbf{E}_Q). Let Q be an algebra query with its set of constructed lineage formulas $\{\varphi^t\}$. If \mathbf{E}_Q gives the generated event relation for Q , the set of all possible answers $Q_{\text{POSS}(\mathcal{W})}$ can be extracted from \mathbf{E}_Q by means of the constructed lineage formulas:

$$Q_{\text{POSS}(\mathcal{W})} = \{t \in \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q) \mid \mathbf{P}(\varphi^t) > 0\}.$$

Proof. See Section (14.1). □

13.3 Second part of \mathbf{E}_Q : all relevant domains \mathbf{D}^t

To prove that an event relation incorporates all relevant domains, we exploit Lemma (12.1) on Page (110).

Lemma 13.3 (\mathbf{E}_Q includes all relevant domains \mathbf{D}^t). Let Q be an algebra query with its equivalent domain calculus query $Q^c = \{\bar{y} \mid \Phi(\bar{y})\}$.

If \mathbf{E}_Q represents the event relation for Q , we can extract the relevant domain \mathbf{D}^t for a specific tuple t as

$$\mathbf{D}^t = \pi_{\text{vars}(\Phi^t)}(\sigma_{(\bar{y}=t)}(\mathbf{E}_Q)).$$

Proof. The proposition directly follows from reusing the construction rules of Lemma (11.3) on Page (106). We only shift the selection of tuples belonging to a specific relevant domain from the start (relation rule in Lemma (11.3) on Page (106)) to the end of our query evaluation. \square

13.4 Third part of \mathbf{E}_Q : all required atomic tuple events

Besides answer tuples and relevant domains, an event relation also collects all atomic tuple events that are needed for constructing the final lineage formulas within our probabilistic query engine. Please recall that our third conceptional transformation step of Section (10.3) made use of binary random variables $X_{R(\bar{c})}$ in order to model atomic tuple events of $(\mathcal{W}, \mathbf{P})$.

Our vertical construction algorithm intensively exploits the relationship between the substituted relation predicates $R(\bar{c})$ of the underlying condition structure ϕ^t and their inferred atomic tuple events used in φ^t .

To provide the connection between relation predicates and atomic tuple events, our relation rule $Q = R$ of Definition (13.1) on Page (124) extends its generated event relation \mathbf{E}_R by a special column storing atomic tuple events.

More concretely, each additional column is labeled by a relation predicate $R(\bar{x}, \bar{c})$ of the underlying condition structure ϕ^t . Such a column then contains all atomic tuple events originating from this specific relation predicate.

A particular atomic tuple event mapped from a relation predicate $R(\bar{x}, \bar{c})$ could be determined by replacing the variables of $R(\bar{x}, \bar{c})$ with the values stored in the same row of \mathbf{E}_Q .

In practice, we do not need to perform any variable substitution for generating an event relation. Instead, we directly use the atomic tuple event labels e_t .

However, we have to be aware of the connection between the atomic tuple events and the relation predicates of the underlying condition structure.

Example 13.2 (Atomic tuple events of event relation \mathbf{E}_Q). *Let us explore the event relation \mathbf{E}_{Q_\setminus} of Figure (13.1) on Page (124). It is built for Q_\setminus with its derived condition structure*

$$\phi_\setminus^t = (R_1(t_{x_A}) \wedge R_3(t_{x_A}, x_B)) \wedge \neg(R_3(t_{x_A}, x'_B) \wedge (t_{x_A} = 1)),$$

see Example (10.7) on Page (90).

Since ϕ_\setminus^t involves the three relation predicates $R_1(t_{x_A})$, $R_3(t_{x_A}, x_B)$, and $R_3(t_{x_A}, x'_B)$, we obtain three atomic tuple events for each row of \mathbf{E}_{Q_\setminus} . For instance, for the first row with $(1, 3, 3)_{(t_{x_A}, x_B, x'_B)}$, we achieve the following three atomic tuple events:

$$\begin{aligned} \{W \in \mathcal{W} \mid X_{R_1(t_{x_A})}(W)\} &= \{W \in \mathcal{W} \mid X_{R_1(1)}(W)\} = e_1 \\ \{W \in \mathcal{W} \mid X_{R_3(t_{x_A}, x_B)}(W)\} &= \{W \in \mathcal{W} \mid X_{R_3(1,3)}(W)\} = e_5 \\ \{W \in \mathcal{W} \mid X_{R_3(t_{x_A}, x'_B)}(W)\} &= \{W \in \mathcal{W} \mid X_{R_3(1,3)}(W)\} = e_5. \end{aligned}$$

Please note that we use the same event label e_5 for

$$\{W \in \mathcal{W} \mid X_{R_3(t_{x_A}, x_B)}(W)\} \quad \text{and} \quad \{W \in \mathcal{W} \mid X_{R_3(t_{x_A}, x'_B)}(W)\},$$

because they both describe the same atomic tuple event

$$\{W \in \mathcal{W} \mid X_{R_3(1,3)}(W)\}.$$

We already showed in Example (13.1) on Page (125) the first incomplete version of our event relations $\mathbf{E}_{Q_{\bowtie}}$, $\mathbf{E}_{Q_{\cup}}$, and $\mathbf{E}_{Q_{\setminus}}$. We purposely left open the encoding of atomic tuple events, which are derived from relation predicates $R(\bar{c}, _)$ with domain values that range over an entire domain. Now, we take care of this particular kind of relation predicates.

In Definition (11.4) on Page (104), we clarified that a relevant domain value $(_)(\bar{x})$ actually expresses its entire domain $\mathbf{D}_{\bar{x}}$. That can lead to an *infinite set* of atomic tuple events.

Example 13.3 (Infinite set of atomic tuple events derived from $R(\bar{c}, _)$). *We continue Example (13.2) on Page (126) by investigating the event relation $\mathbf{E}_{Q_{\setminus}}$ of Figure (13.1) on Page (124) and its underlying condition structure*

$$\phi_{\setminus}^t = (R_1(t_{x_A}) \wedge R_3(t_{x_A}, x_B)) \wedge \neg(R_3(t_{x_A}, x'_B) \wedge (t_{x_A} = 1)).$$

In particular, we are interested in $d_{10} = (5, _)(x_B, x'_B)$ substituting the relation predicate $R_3(t_{x_A}, x'_B)$ of $\phi_{\setminus}^{(2)}$, see fifth row of $\mathbf{E}_{Q_{\setminus}}$ (Figure (13.1) on Page (124)).

If we unfold $d_{10} = (5, _)(x_B, x'_B)$ with $t_{x_A} = (2)$ to capture all encoded atomic tuple events, we obtain the following:

$$\begin{aligned} & \vdots \\ \{W \in \mathcal{W} \mid X_{R_3(2,4)}(W)\} &= \{W \in \mathcal{W} \mid (2, 4) \in R_3(W)\} = \emptyset = F \\ \{W \in \mathcal{W} \mid X_{R_3(2,5)}(W)\} &= e_7 \\ \{W \in \mathcal{W} \mid X_{R_3(2,6)}(W)\} &= e_8 \\ \{W \in \mathcal{W} \mid X_{R_3(2,7)}(W)\} &= \{W \in \mathcal{W} \mid (2, 7) \in R_3(W)\} = \emptyset = F. \\ & \vdots \end{aligned}$$

Additionally, we can provide the infinite set of substituted condition structures implicitly represented by $\phi_{\setminus, \langle \bar{x} \mid d_{10} \rangle}$:

$$\begin{aligned} & \vdots \\ \phi_{\setminus, \langle \bar{x} \mid d'_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 4)} \wedge (2 = 1)) \\ \phi_{\setminus, \langle \bar{x} \mid d''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 5)} \wedge (2 = 1)) \\ \phi_{\setminus, \langle \bar{x} \mid d'''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 6)} \wedge (2 = 1)) \\ \phi_{\setminus, \langle \bar{x} \mid d''''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 7)} \wedge (2 = 1)). \\ & \vdots \end{aligned}$$

These substituted condition structures are derived from the corresponding infinite set of relevant domain values:

$$d_{10} = (5, _)(x_B, x'_B) = \{\dots, \underbrace{(5, 4)}_{d'_{10}}, \underbrace{(5, 5)}_{d''_{10}}, \underbrace{(5, 6)}_{d'''_{10}}, \underbrace{(5, 7)}_{d''''_{10}}, \dots\}.$$

The previous example showed that relation predicates of the form $R(\bar{c}, _)$ imply an infinite list of atomic tuple events.

We cannot insert infinite lists of atomic tuple events into an event relation. Instead, we propose to carry out a simplification of the underlying relevant condition.

In Lemma (14.3) on Page (138), we prove that all relation predicates of the form $R(\bar{c}, _)$ within ϕ^t can be equivalently overwritten by the truth value F , if $\phi_{\langle \bar{x}|d \rangle}^t \equiv T$.

Lemma 13.4 (Simplifying relation predicates of the form $R(\bar{c}, _)$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}^t} \phi_{\langle \bar{x}|d \rangle}^t\}.$$

Then,

$$(\phi_{\langle \bar{x}|d \rangle}^t \equiv T) \Rightarrow ((\phi_{\langle \bar{x}|d \rangle}^t)_{\langle R(\bar{c}, _)|F \rangle} \equiv T)$$

holds.

Proof. See Section (14.2). □

Example 13.4 (Relation predicates $R(\bar{c}, _)$ simplified to F). *Again, we examine the condition structure $\phi_{\setminus, \langle \bar{x}|d_{10} \rangle}^t$, which involves the relation predicate $R_3(5, _)$, see Example (13.3) on Page (127). Specifically, we verify that*

$$\begin{aligned} \phi_{\setminus, \langle x_B, x'_B | d_{10} \rangle}^t &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, _) \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)) \\ &= (\phi_{\setminus, \langle x_B, x'_B | d_{10} \rangle}^t)_{\langle R_3(2, _)|F \rangle} \end{aligned}$$

holds, if we evaluate the corresponding list of condition structures

$$\dots, \quad \phi_{\setminus, \langle x_B, x'_B | d'_{10} \rangle}^t, \quad \phi_{\setminus, \langle x_B, x'_B | d''_{10} \rangle}^t, \quad \phi_{\setminus, \langle x_B, x'_B | d'''_{10} \rangle}^t, \quad \phi_{\setminus, \langle x_B, x'_B | d''''_{10} \rangle}^t, \quad \dots$$

in the world W^{max} of Example (4.1) on Page (24).

In this case, our proposed simplification assures that we can overwrite the original truth values of all relation predicates derived from $R_3(2, _)$ with F :

$$\begin{aligned} &\vdots \\ \phi_{\setminus, \langle x_B, x'_B | d'_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 4)} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{F} \wedge (2 = 1)) \\ &\equiv T \\ \phi_{\setminus, \langle x_B, x'_B | d''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 5)} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{T} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{F} \wedge (2 = 1)) \\ &\equiv T \\ \phi_{\setminus, \langle x_B, x'_B | d'''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 6)} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{T} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{F} \wedge (2 = 1)) \\ &\equiv T \\ \phi_{\setminus, \langle x_B, x'_B | d''''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{R_3(2, 7)} \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(\underline{F} \wedge (2 = 1)) \end{aligned}$$

$\mathbf{E}_{Q_{\bowtie}}$						
	t_{x_A}	x_B	$R_1(x_A)$	$R_2(x_A, x_B)$	$R_3(x_A, x_B)$	
d_1	1	3	e_1	e_3	e_5	
d_2	1	4	e_1	e_4	e_6	

$\mathbf{E}_{Q_{\cup}}$						
	t_{x_A}	x_B	x'_B	$R_2(x_A, x_B)$	$R_1(x_A)$	$R_2(x_A, x'_B)$
d_3	1	4	3	e_4	e_1	F
d_4	1	—	3	F	e_1	e_3
d_5	1	—	4	F	e_1	e_4

$\mathbf{E}_{Q_{\setminus}}$						
	t_{x_A}	x_B	x'_B	$R_1(x_A)$	$R_3(x_A, x_B)$	$R_3(x_A, x'_B)$
d_6	1	3	3	e_1	e_5	e_5
d_7	1	3	4	e_1	e_5	e_6
d_8	1	4	3	e_1	e_6	e_5
d_9	1	4	4	e_1	e_6	e_6
d_{10}	2	5	—	e_2	e_7	F
d_{11}	2	6	—	e_2	e_8	F

Figure 13.2: Final event relations generated as inputs for our vertical lineage construction algorithm

$$\equiv T.$$

$$\vdots$$

Remarkably, the overall truth values $\phi_{\setminus, \langle \bar{x} | d_{10}' \rangle}^{(2)} \equiv T$ and $\phi_{\setminus, \langle \bar{x} | d_{10}'' \rangle}^{(2)} \equiv T$ are still valid, despite the relation predicates $R_3(2, 5)$ and $R_3(2, 6)$ have been replaced by their opposite truth value F.

Lemma (14.3) on Page (138) permits us to simply put the impossible tuple event F into our event relation for atomic tuple events based on $R(\bar{c}, _)$ -predicates.

Example 13.5 (Final event relations). *In Figure (13.2) on Page (129), we eventually present our final event relations \mathbf{E}_{\bowtie} , \mathbf{E}_{\cup} , and \mathbf{E}_{\setminus} . They represent the outcome of our relational database layer as well as the input for our vertical lineage construction algorithm.*

Our simplification rule significantly contributes to the flexibility that characterizes our approach.

Since \mathbf{E}_Q collects all required atomic tuple events without any pre-fixed lineage structure, our algorithm has the freedom of (re)arranging atomic tuple events on the most fine-grained level in order to build our alternative lineage formulas.

Remark 13.1 (Relevant domains vs. U-relations (MayBMS)). *When we compare our event relations with U-relations as proposed in [7], we recognize that both approaches construct identical structured relations for queries from SPJ. In other words, for SPJ-queries our event relation equals a relation that contains a set of lineage formulas in DNF.*

U-relation for Q_{\bowtie}					U-relation for Q_{\cup}			
A	B	E_{R_1}	E_{R_2}	E_{R_3}	A	B	E_{R_2}	E_{R_1}
1	3	e_1	e_3	e_5	1	4	e_4	T
1	4	e_1	e_4	e_6	1	3	e_3	e_1
					1	4	e_4	e_1

Figure 13.3: U-relations for Q_{\bowtie} and Q_{\cup}

We emphasize that this correspondence only holds for the query class **SPJ**. In general, our approach provides full algebra support and is far away from constructing lineage formulas in DNF. A relation storing a conjunct per row would have been up to 2^n tuples in general, where n describes the given database size.

To underline the divergence between event relations and U-relations, we additionally show in Figure (13.3) on Page (130) the U-relations for our queries Q_{\bowtie} and Q_{\cup} . Please remember that there is no U-relation defined for our third query Q_{\setminus} , since U-relations forbid difference operations [7].

13.5 Experiments: relational processing

In the following, we present the results of our first series of experiments aimed at investigating the query processing part within a relational database layer. We describe the following points in more detail:

- the five tested basic approaches,
- implementation details of our investigated basic approaches,
- the data we measured, and
- our experimental observations.

Section (A.1) provides a comprehensive introduction to our underlying test environment. It comprises two probabilistic databases with two corresponding query sets that have been constructed from two public available data sets. We conducted all our current and following sets of experiments by means of two test databases, namely:

- a TID version based on the IMDB movie database¹ and
- a BID variant of the TPC-H benchmark database².

To be more precise, we evaluated the following six IMDB queries (in abbreviated form):

$$\begin{aligned}
\text{IMDB-Q1} &= \pi_{B,C}((R_1(A,B) \bowtie R_2(A)) \bowtie R_3(A,C)) \\
\text{IMDB-Q2} &= \pi_{A,B}(((R_1(A,B) \bowtie R'_1(B)) \bowtie R_3(A)) \bowtie R_4(A)) \\
\text{IMDB-Q3} &= \pi_A(((R_1(A) \bowtie R_2(A)) \bowtie R_3(A)) \bowtie R_4(A)) \\
\text{IMDB-Q4} &= (R_1(A) \bowtie R_2(A)) \cup \pi_A((R'_1(A) \bowtie R_3(A,B)) \bowtie R_4(B)) \\
\text{IMDB-Q5} &= ((R_1(A) \bowtie R_2(A)) \bowtie R_3(A,B)) \setminus \pi_{A,B}((R_4(A,B,C) \bowtie R_5(C)) \bowtie R_6(A)) \\
\text{IMDB-Q6} &= ((R_1(A) \cup R_2(A)) \setminus R_3(A)) \bowtie \pi_A((R_4(A,B) \bowtie R_5(B)) \bowtie R_6(A))
\end{aligned}$$

and seven TPC-H queries (in abbreviated form):

$$\begin{aligned}
\text{TPCH-Q1} &= \pi_{B,C,D}(R_1(A) \bowtie R_2(A,B,C,D)) \bowtie R_3(B) \\
\text{TPCH-Q2} &= \pi_{A,C,\dots,H}(R_4(A,B) \bowtie R_1(B,C,\dots,H)) \bowtie \pi_C(R_2(C,I) \bowtie R_3(I))
\end{aligned}$$

¹<http://www.imdb.com>

²<http://www.tpc.org/tpch>

$$\begin{aligned}
\text{TPCH-Q3} &= (\pi_B(R_1(A) \bowtie R_2(A, B)) \bowtie R_3(B, C)) \bowtie \pi_C(R_4(D) \bowtie R_5(C, D)) \\
\text{TPCH-Q4} &= \pi_B((R_1(A) \bowtie R_2(A, B)) \bowtie R_3(A)) \bowtie \pi_B(R_4(C) \bowtie R_5(B, C)) \\
\text{TPCH-Q5} &= \pi_B(R_1(A) \bowtie R_3(A, B) \bowtie R_2(B)) \cup \pi_B(R_4(B, C) \bowtie R_5(C)) \\
\text{TPCH-Q6} &= R_1(A) \bowtie (\pi_B(R_2(A, B) \bowtie R_3(A)) \setminus \pi_B(R_4(C, B) \bowtie R_5(C))) \\
\text{TPCH-Q7} &= \pi_A(R_1(B) \bowtie R_2(A, B)) \setminus (\pi_A(R_3(C) \bowtie R_4(A, C)) \cup \pi_A(R_5(A, D) \bowtie R_6(D))).
\end{aligned}$$

All these queries are discussed in more depth in Section (A.2).

Investigated basic approaches

In the four series of experiments presented in Part (V), we examined the performances of the following five basic techniques:

- **safe plans:** computation of answer probabilities within an RDBMS by means of *safe plans*, as proposed by MystiQ [94],
- **DNF lineage:** generation of *lineage formulas in DNF*, as used in MayBMS [7], SPROUT [83] and ProQua [70],
- **nested lineage:** classical construction of *nested lineage formulas*, as conceptionally proposed by Fuhr and Rölleke [39] and practically implemented within SPROUT2 [37],
- **networks:** creation of *factors and Boolean functions networks*, as suggested by PrDB [101] and Trio [28],
- **vertical:** *vertical lineage construction* based on *event relations* in composed and decomposed form, as developed for Prophecy.

Please recall that we already outlined the key ideas of the first four basic approaches in Chapter (8). In addition, we next describe particular implementation characteristics for all tested basic approaches.

Relational processing: safe plans

As mentioned earlier, safe plans were originally developed for the MystiQ system [94]. In contrast to others systems, MystiQ does not employ an additional query engine in addition to an RDBMS. The entire computation of answer probabilities is pushed into the relational database layer.

Following [94], we implemented the required probability aggregation rules of Lemma (6.3) on Page (40) as user-defined aggregation functions. The safe forms of our tested queries (Section (A.2)) guaranteed that only probabilities of independent or mutually exclusive atomic tuple events got combined.

Since all answer probabilities were computed within the relational database layer, a further loading step into a probabilistic query engine followed by an additional optimization/evaluation phase was not necessary any more.

Relational processing: lineage formulas in DNF

Our second tested approach constructed lineage formulas in DNF. Antova et al. developed this method known as U-relations for their MayBMS system [7, 60].

In general, U-relations provide the input for the probabilistic query engine SPROUT [83, 82]. In a specific U-relation, each row carries exactly one conjunct of a contained lineage formula in DNF. An atomic tuple event of a specific conjunct/row is thereby stored in a specific attribute. An example of a U-relation is given in Figure (8.3) on Page (56).

Lineage formulas in DNF can be easily generated for **SPJ**-queries. To do so, we can make use of the classical construction rules, when all event columns are collected separately and all projections are carried out as last operations [7].

We again emphasize that U-relations only support **SPJ**-queries [7], i.e., IMDB-Q1, . . . , IMDB-Q3, and TPCH-Q1, . . . , TPCH-Q4. Difference operations, for example, are not allowed, since they imply negated lineage subformulas, the unfolding of which would be very costly.

Relational processing: nested lineage formulas

In contrast to U-relations, SPROUT2 by Fink, Olteanu, and Rath facilitates queries with difference operations [37]. By exploiting the classical construction rules of Fuhr and Röllecke (Lemma (6.1) on Page (35)) each row of their computed relations contains one answer tuple extended by its nested lineage formula. It is stored in a dedicated attribute field shown in Figure (8.2) on Page (55).

Please recall from Lemma (6.2) on Page (37) that the size of a lineage formula generally grows polynomially in input size. In theory, the length of only one attribute value of one resulting row can already exceed the size of the given input database. In practice, the limited attribute sizes of an RDBMS inhibit such a behaviour. However, this also means that the length of a lineage formula is limited in advance.

In our experiments, we implemented several user-defined functions within the relational database layer for creating and storing nested lineage formulas as JSON data objects in CLOB attribute fields. In order to set up tractable lineage formulas, we employed the safe forms of our tested queries given in Figure (A.5) and (A.6) on Page (250) and (251).

Relational processing: factor networks

In Chapter (8), we demonstrated how the probabilistic database systems PrDB and Trio set up networks of factors and Boolean functions. Both network types can be easily interpreted as lineage formulas.

To create their networks, PrDB and Trio use rules conceptionally similar to the ones presented in Lemma (6.1) on Page (35). They generate one tuple for each lineage *subformula* appearing during the overall lineage construction process.

Relational processing: vertical lineage construction

In the first sections of that chapter, we exhaustively introduced the key ideas behind our concept of event relations. Again, we underline that an event relation does not reflect any structures of our final lineage formulas. Among other advantages, this gives us the freedom of generating an event relation in different forms. We only have to ensure that all three necessary input parts can be still delivered.

Besides the generation of the standard form of an event relation (Definition (13.1) on Page (124)) our experiments additionally exploited a second variant of event relations derived from the well-known concept of relation decomposition. A detailed discussion of that technique is provided as an advanced topic in Chapter (14.3). Specifically, we generated composed or decomposed event relations for the following queries:

- Composed event relations:
 - IMDB queries: IMDB-Q4, IMDB-Q5, and IMDB-Q6
 - TPC-H queries: TPC-H-Q1 and TPC-H-Q2
- Decomposed event relations:
 - IMDB queries: IMDB-Q1, IMDB-Q2, and IMDB-Q3
 - TPC-H queries: TPC-H-Q3, TPC-H-Q4, TPC-H-Q5, TPC-H-Q6, and TPC-H-Q7.

Remark 13.2 (Decomposition of U-relations). *A decomposition of U-relations is not reasonable, since their evaluation algorithms SPROUT and D-trees insist on lineage formulas in DNF as inputs. In a decomposed form, all inputs had to be joined again before they can be processed. Since join operations are highly optimized within an RDBMS, we avoid to delegate this kind of operations to a probabilistic query engine.*

To the contrary, our vertical construction algorithm can be easily adapted to process decomposed event relations without any further expensive join/unfolding steps, see Chapter (14.3) and (16.2).

Measured properties: relational processing						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	RDBMS relation(s)	1	1	1	13	7
	RDBMS tuples	5,473	91,406	5473	369,212	5,480
IMDB-Q2	RDBMS relation(s)	1	1	1	7	5
	RDBMS tuples	30	106,033	30	37,298	3,400
IMDB-Q3	RDBMS relation(s)	1	1	1	8	5
	RDBMS tuples	1,516	940,008	1,516	215,636	1,520
IMDB-Q4	RDBMS relation(s)	1	-	1	10	1
	RDBMS tuples	2,216	-	2,216	926,081	2,995
IMDB-Q5	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	5,535	-	5,535	1,744,059	6,257
IMDB-Q6	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	141	-	141	655,243	1,480

Measured computation times: relational processing						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	relational	10.36	1.612	16.806	3.301	1.385
	total	10.36	1.612	16.806	3.301	1.385
IMDB-Q2	relational	1.109	0.428	17.971	0.368	0.267
	total	1.109	0.428	17.971	0.368	0.267
IMDB-Q3	relational	33.898	3.958	190.858	7.075	2.839
	total	33.898	3.958	190.858	7.075	2.839
IMDB-Q4	relational	13.31	-	14.77	3.699	0.097
	total	13.31	-	14.77	3.699	0.097
IMDB-Q5	relational	32.182	-	52.796	15.478	0.593
	total	32.182	-	52.796	15.478	0.593
IMDB-Q6	relational	14.48	-	19.746	5.375	0.455
	total	14.48	-	19.746	5.375	0.455

Figure 13.4: IDMB queries: measured properties and computation times of relational database layer experiments

Measured data

Figure (13.4) and (13.5) on Page (136) and (137) show the results of our first series of experiments:

- the number of all resulting relations per approach,
- the accumulated tuple counts of all generated relations per approach, and
- the mean of the processing times (wall-clock) of 100 runs for generating all resulting relations per approach.

To elucidate the effects of the current and coming evaluation phases, we split the total processing times into partial times. They are associated with our different processing parts, namely relational processing, lineage construction, probability computation, and lineage compression. In the coming series of experiments, we add further components to the total times.

Experimental observations

Next, we discuss the properties and computation times measured for our first series of experiments.

Measured properties: RDBMS relation(s) and tuples

The number of RDBMS relations and tuples reflects the output sizes of the relational database layer and the input sizes of the probabilistic query engine.

Measured properties: relational processing						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	RDBMS relation(s)	1	1	1	6	1
	RDBMS tuples	125	492	125	154,047	492
TPCH-Q2	RDBMS relation(s)	1	1	1	10	1
	RDBMS tuples	961	1,475	961	176,761	1,475
TPCH-Q3	RDBMS relation(s)	1	1	1	12	6
	RDBMS tuples	38	890,072	38	239,647	11,453
TPCH-Q4	RDBMS relation(s)	1	1	1	11	6
	RDBMS tuples	14	1,277,028	14	361,648	1,125
TPCH-Q5	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	99	-	99	151,520	1,520
TPCH-Q6	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	63	-	63	348,128	5,045
TPCH-Q7	RDBMS relation(s)	1	-	1	14	7
	RDBMS tuples	27	-	27	493,531	2,166

Measured computation times: relational processing						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	relational	12.332	0.112	15.082	2.173	0.113
	total	12.332	0.112	15.082	2.173	0.113
TPCH-Q2	relational	8.296	0.489	10.96	2.353	0.488
	total	8.296	0.489	10.96	2.353	0.488
TPCH-Q3	relational	7.363	2.411	12.847	3.721	1.683
	total	7.363	2.411	12.847	3.721	1.683
TPCH-Q4	relational	14.536	2.649	55.117	5.88	1.207
	total	14.536	2.649	55.117	5.88	1.207
TPCH-Q5	relational	1.877	-	4.674	0.81	0.37
	total	1.877	-	4.674	0.81	0.37
TPCH-Q6	relational	5.675	-	12.307	3.65	0.898
	total	5.675	-	12.307	3.65	0.898
TPCH-Q7	relational	9.534	-	15.386	3.862	1.381
	total	9.534	-	15.386	3.862	1.381

Figure 13.5: TPC-H queries: measured properties and computation times of relational database layer experiments

- **safe plans**: Safe plans created one resulting relation containing all possible answers with their corresponding answer probabilities.
- **DNF lineage**: Lineage formulas in DNF were built in one relation, where each tuple represents one conjunct. The number of all unfolded conjuncts was usually larger than the number of possible answers and the number of all nested lineage subformulas.
- **nested lineage**: Every nested lineage formula was constructed in one dedicated attribute field. Accordingly, we achieved one resulting relation storing an extended tuple for each possible answer.
- **networks**: Networks of factors were built in different relations based on all subqueries of the given input query. Thereby, each tuple represented a lineage subformula occurring during the evaluation process. This explains the relatively large number of resulting tuples.
- **vertical**: Event relations were generated in composed and decomposed form. Accordingly, we obtained one resulting relation in the composed case and a set of resulting relations comprising one join and several event source relations in the decomposed case. The tuple numbers of composed event relations for **SPJ**-queries equaled the DNF lineage approach, see **SPJ**-queries TPCH-Q1 and TPCH-Q2. However, the benefits of decomposing our event relations in comparison to DNF lineage were clearly visible, see **SPJ**-queries IMDB-Q1, . . . ,

IMDB-Q3, TPC-H-Q3, and TPC-H-Q4.

Measured computation times: relational processing part

- **safe plans:** Safe plans generated the smallest resulting sets. Yet, their computation times were clearly slower than the times observed for the DNF lineage, nested lineage, and vertical approach, since safe plans already included the calculation of all answer probabilities performed by specialised aggregation functions.
- **DNF lineage:** The DNF lineage technique was able to compute their resulting sets very quickly, because there were no expensive projection operations involved. Moreover, the join orders have been chosen by the underlying RDBMS optimizer.
- **nested lineage:** The generation of nested lineage formulas within large CLOB fields were showed the worst computation times by far.
- **networks:** Since the resulting relations of our factor networks were built just using native relational operators and data types, their computation times were clearly better than the cases of safe plans and nested lineage.
- **vertical:** The obtained computation times for event relations were the fastest. Our method could exploit decomposition and native operators/data types without using any expensive projection operations and pre-defined join orders.

13.6 Summary

In this chapter, we further developed the groundwork already carried out in Chapter (11). So, we developed a set of efficient and simple generator rules for event relations on the basis of the rules presented in Chapter (11). Event relations represent one of the two main inputs for our vertical lineage construction algorithm. They deliver the set of all possible answers, all relevant domains, and all involved atomic tuple events. Moreover, we presented our first series of experiments dedicated to the query processing part of the relational database layer.

Chapter 14

Advanced aspects of event relations

In this chapter, we discuss in this chapter a handful advanced properties and aspects of event relations:

- Section (14.1): the containment of all possible answers $Q_{\text{poss}(\mathcal{W})}$ in \mathbf{E}_Q ,
- Section (14.2): our simplification rule for rewriting $R(\bar{c}, _)$ to \mathbb{F} , and
- Section (14.3): event relations in a decomposed form.

14.1 Containment of all possible answers $Q_{\text{poss}(\mathcal{W})}$ in \mathbf{E}_Q

Next, we outline the two missing proofs of Section (13.2). They assure that an event relation includes all possible tuple answers $Q_{\text{poss}(\mathcal{W})}$.

Lemma 14.1 (\mathbf{E}_Q includes the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$). *Let Q be an algebra query and let \mathbf{E}_Q be its event relation. Then, \mathbf{E}_Q can be used to extract the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$:*

$$Q_{\text{poss}(\mathcal{W})} \subseteq \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q).$$

Proof. The core idea of our proof is to give rules for constructing a superset of $Q_{\text{poss}(\mathcal{W})}$, which are clearly covered by our rules of Definition (13.1) on Page (124). Accordingly, our rules of Definition (13.1) on Page (124) computes a superset of $Q_{\text{poss}(\mathcal{W})}$, which proves our proposition.

First, we consider a query Q from **SPJU**. In this case, we compute $Q_{\text{poss}(\mathcal{W})}$ by recursively applying following rules:

$$\begin{aligned} Q = R & : Q_{\text{poss}(\mathcal{W})} := R(W^{\text{max}}) \\ Q = \sigma_F(Q_1) & : Q_{\text{poss}(\mathcal{W})} := \sigma_F(Q_{1,\text{poss}(\mathcal{W})}) \\ Q = \pi_{\mathcal{A}}(Q_1) & : Q_{\text{poss}(\mathcal{W})} := \pi_{\mathcal{A}}(Q_{1,\text{poss}(\mathcal{W})}) \\ Q = Q_1 \bowtie Q_2 & : Q_{\text{poss}(\mathcal{W})} := Q_{1,\text{poss}(\mathcal{W})} \bowtie Q_{2,\text{poss}(\mathcal{W})} \\ Q = Q_1 \cup Q_2 & : Q_{\text{poss}(\mathcal{W})} := Q_{1,\text{poss}(\mathcal{W})} \cup Q_{2,\text{poss}(\mathcal{W})} \\ Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) & : Q_{\text{poss}(\mathcal{W})} := \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_{1,\text{poss}(\mathcal{W})}). \end{aligned}$$

It is clear that we only transfer the algebra operations of Q to the possible answer sets to be built. Because we start with the maximal possible sets of tuples for all relations, we always achieve the maximal possible set of tuples for an **SPJU**-(sub)query. This directly follows from the semantics of the used algebra operators.

This does not longer hold for queries with difference operations. For instance, the maximal possible set of tuples for an **nrSPJUD**-query $Q \equiv Q_1 \setminus Q_2$ is given, when $Q_2(W)$ is minimal instead of maximal, i.e.,

$$Q(W^{\text{max}}) := (Q_1(W^{\text{max}}) \setminus Q_2(W^{\emptyset})) = (Q_1(W^{\text{max}}) \setminus \emptyset) = Q_1(W^{\text{max}}).$$

As a consequence, we introduce a difference rule for computing $Q_{\text{poss}(\mathcal{W})}$ in form of

$$Q \equiv Q_1 \setminus Q_2 \quad : \quad Q_{\text{poss}(\mathcal{W})} := Q_{1, \text{poss}(\mathcal{W})},$$

which simply sets $Q_{2, \text{poss}(\mathcal{W})}$ to the empty set.

Please be aware that our difference rule only generates the exact maximal possible set of tuples, if all possible answer sets of Q_1 and Q_2 are disjoint. In the case of repeating queries, this assumption does not hold anymore. For example, the set of all possible answers for the query $Q \equiv Q_1 \setminus Q_1$ is obviously empty instead of $Q_{1, \text{poss}(\mathcal{W})}$ produced by our difference rule. However, our difference rule at least always gives us a superset of $Q_{\text{poss}(\mathcal{W})}$.

If we finally compare our seven rules for creating a superset of $Q_{\text{poss}(\mathcal{W})}$ with our rules for generating the corresponding event relation one-by-one, we can immediately verify that for each rule

$$Q_{\text{poss}(\mathcal{W})} \subseteq \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q)$$

is valid. □

In the last lemma, we confirmed that \mathbf{E}_Q includes $Q_{\text{poss}(\mathcal{W})}$ in the form of a superset. To get the exact set $Q_{\text{poss}(\mathcal{W})}$, we have to filter the candidates given in \mathbf{E}_Q . For that task, we take advantage of the probabilities of our final lineage formulas $\mathbf{P}(\varphi^t)$.

Lemma 14.2 (Extracting $Q_{\text{poss}(\mathcal{W})}$ from \mathbf{E}_Q). *Let Q be an algebra query with its set of constructed lineage formulas $\{\varphi^t\}$. If \mathbf{E}_Q gives the generated event relation for Q , the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$ can be extracted from \mathbf{E}_Q by means of the constructed lineage formulas:*

$$Q_{\text{poss}(\mathcal{W})} = \{t \in \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q) \mid \mathbf{P}(\varphi^t) > 0\}.$$

Proof.

$$\begin{aligned} \mathbf{P}(\varphi^t) > 0 &\Leftrightarrow \mathbf{P}(\Phi_{\langle \bar{x} \rangle \mathbf{D}^t}^t) > 0 && \text{(Theorem (10.1))} \\ &\Leftrightarrow \text{satWorlds}(\Phi_{\langle \bar{x} \rangle \mathbf{D}^t}^t) \neq \emptyset && \text{(Definition (11.1))} \\ &\Leftrightarrow \exists W \in \mathcal{W} : t \in Q^c(W) \\ &\Leftrightarrow \exists W \in \mathcal{W} : t \in Q(W) && \text{(Lemma (10.1))} \\ &\Leftrightarrow t \in Q_{\text{poss}(\mathcal{W})}. \end{aligned}$$

□

14.2 Simplifying relation predicates of the form $R(\bar{c}, _)$

In Section (13.4), we explained that an event relation contains all atomic tuple events that are required for our final lineage formulas. More specifically, we store all atomic tuple events, which originate from the relation predicates $R(\bar{x}, \bar{c})$ of the underlying calculus condition ϕ^t .

As shown in Example (13.3) on Page (127), the decoding of atomic tuple events derived from relation predicates $R(\bar{c}, _)$ leads to infinite set of atomic tuple events. To avoid such sets, we prove that relation predicates of the form $R(\bar{c}, _)$ can be simplified to the truth value F. Then, we simply have to store the impossible event F as an atomic tuple event.

Lemma 14.3 (Simplifying relation predicates of the form $R(\bar{c}, _)$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \overline{\bigvee_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x} \mid d \rangle}^t\}.$$

Then, we can equivalently replace all relation predicates $R(\bar{c}, _)$ in $\phi_{\langle \bar{x} \mid d \rangle}^t$ with F , given that $\phi_{\langle \bar{x} \mid d \rangle}^t \equiv T$, i.e.,

$$(\phi_{\langle \bar{x} \mid d \rangle}^t \equiv T) \Rightarrow ((\phi_{\langle \bar{x} \mid d \rangle}^t)_{\langle R(\bar{c}, _) \mid F \rangle} \equiv T)$$

evaluated in an arbitrary world W of \mathcal{W} .

Proof. Definition (11.4) on Page (104) shows that a domain value $d = (\bar{c}, _)$ can only occur during a union or a difference operation, i.e., $(Q_1 \cup Q_2)$ or $(Q_1 \setminus Q_2)$. In the following, we discuss these both operators as Case (A) and Case (B). Thereby, we assume that the proposition already holds for condition structures derived from the subqueries Q_1 and Q_2 of $(Q_1 \cup Q_2)$ or $(Q_1 \setminus Q_2)$. Accordingly, we implicitly use an induction proof over the structure of Q . In contrast to other induction proofs in this work, we can focus here on the operators \cup and \setminus , since they are the only ones yielding $d = (\bar{c}, _)$.

Additionally, we introduce two variable sets

$$\bar{x}_1 := \text{vars}(\text{head}(Q_1)) \quad \text{and} \quad \bar{x}_2 := \text{vars}(\text{head}(Q_2)).$$

They describe the variables for the output attributes of our subqueries Q_1 and Q_2 .

Case (A): union operator $Q = Q_1 \cup Q_2$:

The condition structure implied by a union operation is given as

$$\phi_{Q, \langle \bar{x} \mid d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 \mid d_{\bar{x}_1} \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t,$$

see MAC rules of Lemma (10.1) on Page (84). If we take our general assumption $\phi_{Q, \langle \bar{x} \mid d \rangle}^t \equiv T$ into account, we know that $\phi_{Q_1, \langle \bar{x}_1 \mid d_{\bar{x}_1} \rangle}^t$ and/or $\phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t$ have to be fulfilled.

Without losing generality, we assume $\phi_{Q_1, \langle \bar{x}_1 \mid d_{\bar{x}_1} \rangle}^t \equiv T$. Then, $d_{\bar{x}_2}$ of $(d_{\bar{x}_1} \bullet d_{\bar{x}_2}) \in \mathbf{D}^t$ can range over its entire domain $\mathbf{D}_{\bar{x}_2}$, see Definition (12.1) on Page (109). We express this property by relation predicates of the form $R(\bar{c}, _)$ in $\phi_{Q, \langle \bar{x} \mid d \rangle}^t$.

In general, we do not know the truth value of a single relation predicate, if it is (partially) based on relevant domain values, which range over an entire domain. Nevertheless, we can replace such substituted relation predicates by F , because the first operand $\phi_{Q_1, \langle \bar{x}_1 \mid d_{\bar{x}_1} \rangle}^t \equiv T$ always ensures the overall truth value T . So, we can assign F to all relation predicates of the form $R(\bar{c}, _)$ in the second operand $\phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t$ regardless of how this replacement effects the truth value of $\phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t$.

Case (B): difference operator $Q = Q_1 \setminus Q_2$:

The MAC rules of Lemma (10.1) on Page (84) give us the following condition structure for a difference operation:

$$\phi_{Q, \langle \bar{x} \mid d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 \mid d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t).$$

Moreover, the difference rule of Definition (12.1) on Page (109) implies that relevant domain values, which range over an entire domain can only be created for $d_{\bar{x}_2}$ of $(d_{\bar{x}_1} \bullet d_{\bar{x}_2}) \in \mathbf{D}^t$.

Furthermore, we can be sure that the second subcondition structure $\phi_{Q_2, \langle \bar{x}_2 \mid d_{\bar{x}_2} \rangle}^t$ has to equal F , if $\phi_{Q, \langle \bar{x} \mid d \rangle}^t \equiv T$ is given.

By combining the last two facts, we need to prove that

$$\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t) \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \underbrace{\neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)}_{\mathbf{F}} \langle R(\bar{c}, _) | \mathbf{F} \rangle \equiv \mathbf{T},$$

if $\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \mathbf{T}$. In order to do so, we use an induction proof over the number of relation predicates of the form $R(\bar{c}, _)$ involved in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$.

I.) Induction basis ($n = 1$ relation predicate $R(\bar{c}, _)$ in $\langle \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \mathbf{F} \rangle$:

Since we assume that there is only one relation predicate of the form $R(\bar{c}, _)$ in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \mathbf{F}$, we can identify six different syntactic variants of $R(\bar{c}, _)$ appearing in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$.

Those variants are obviously related to the MAC rule of Lemma (10.1) on Page (84), because the condition structure $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ is derived from the algebra subquery Q_2 of the original query $Q \equiv (Q_1 \setminus Q_2)$ considered in Case (B).

In all following subcases, we can be sure that there is no further relation predicate of the form $R(\bar{c}, _)$ in the subcondition structure $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t$, since there is only one $R(\bar{c}, _)$ -predicate in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$. The goal of all of our six cases is to prove that the truth value \mathbf{F} of $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ is preserved by replacing $R(\bar{c}, _)$ with \mathbf{F} .

Case $\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle$ derived from $Q_2 = R$:

We can replace $R(\bar{c}, _)$ directly and achieve the following:

$$\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \equiv \mathbf{F}.$$

Case $\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \wedge F$ derived from $Q_2 = \sigma_F(R)$:

Again, we can rewrite $R(\bar{c}, _)$ directly and see that

$$\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \wedge F \equiv \mathbf{F} \wedge F \equiv \mathbf{F}.$$

Case $\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle$ derived from $Q_2 = Q_3 \bowtie R$:

When we set $R(\bar{c}, _)$ to \mathbf{F} , we obtain:

$$\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \mathbf{F} \equiv \mathbf{F}.$$

Case $\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \vee R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle$ derived from $Q_2 = Q_3 \cup R$:

In order to fulfill the assumption $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \mathbf{F}$, both operands of $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ must equal \mathbf{F} . This leads to the following equivalence:

$$\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \vee R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \equiv \mathbf{F} \vee \mathbf{F} \equiv \mathbf{F}.$$

Case $\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \wedge \neg(\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)$ derived from $Q_2 = R \setminus Q_3$:

By simplifying $R(\bar{c}, _)$ to \mathbf{F} , we obtain the following:

$$\mathbf{F} \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv R(\bar{c}, _) \langle R(\bar{c}, _) | \mathbf{F} \rangle \wedge \neg(\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t) \equiv \mathbf{F} \wedge \neg(\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t) \equiv \mathbf{F}.$$

Case $F \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(R(\bar{c}, _)_{\langle R(\bar{c}, _) | F \rangle})$ derived from $Q_2 = Q_3 \setminus R$:

This case is more complex. Our idea is to exploit the connection between the negated relation predicate $\neg(R(\bar{c}, _))$ and its constructed relevant domain of Definition (12.1) on Page (109). By applying our difference rule of Definition (12.1) on Page (109), i.e.,

$$\begin{aligned} Q_2 \equiv Q_3 \setminus R & : \mathbf{D}_{Q_2}^t := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ \mathcal{M}_1 & := \sigma_{(d \text{ is relevant})}(\mathbf{D}_{Q_3}^{t_3} \bowtie \mathbf{D}_R^t) \\ \mathcal{M}_2 & := (\mathbf{D}_{Q_3}^t \setminus \pi_{\text{attr}(\mathbf{D}_{Q_3}^t)}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_2}\}, \end{aligned}$$

we see that $\neg(R(\bar{c}, _))$ always comes into play, if we generate the relevant domain value set \mathcal{M}_2 .

Then, our difference rule of Lemma (12.1) on Page (110) states that the variables $\bar{y} \in \mathbf{D}_{Q_3}^t$ can range in $\neg(R(\bar{c}, \bar{y})) = \neg(R(\bar{c}, _))$ over their whole domain (i.e., $\mathbf{D}_{\bar{y}} = \mathbf{D}_{Q_3}^t$) with $\neg(R(\bar{c}, \bar{y})) \equiv \neg(F) \equiv \top$. Accordingly, no domain value (\bar{c}, \bar{y}) is relevant for $R(\bar{c}, \bar{y})$. Then, the first operand $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t$ of $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ must always equal F , if the assumption $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv F$ holds. As a result, we can rewrite $R(\bar{c}, _)$ as follows:

$$F \equiv \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(R(\bar{c}, _)_{\langle R(\bar{c}, _) | F \rangle}) \equiv F \wedge \neg(F) \equiv F.$$

II.) Induction assumption (n relation predicates $R(\bar{c}, _)$ in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv F$):

The property

$$\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t) \equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \equiv \top$$

holds for all subcondition structures $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ with n or less relation predicates of the form $R(\bar{c}, _)$. In conjunction with Case (A), we can additionally manifest the main proposition of this lemma

$$(\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \top) \Rightarrow ((\phi_{Q, \langle \bar{x} | d \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \equiv \top)$$

for all condition structures $\phi_{Q, \langle \bar{x} | d \rangle}^t$ with n or less relation predicates of the form $R(\bar{c}, _)$. We denote the first and second part of our induction assumption as (IA1) and (IA2).

III.) Induction step ($(n+1)$ relation predicates $R(\bar{c}, _)$ in $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$):

We again prove that the truth value of $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv F$ does not change after performing the additional substitution $(\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle}$. Firstly, we note that a subformula with $(n+1)$ relation predicates of the form $R(\bar{c}, _)$ can only emerge in the following three condition structures:

$$\begin{aligned} \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t & \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t \equiv F \\ \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t & \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \vee \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t \equiv F \\ \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t & \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t) \equiv F. \end{aligned}$$

They are obviously inferred from the three algebra subqueries

$$Q_2 = (Q_3 \bowtie Q_4), \quad Q_2 = (Q_3 \cup Q_4), \quad \text{and} \quad Q_2 = (Q_3 \setminus Q_4).$$

In each case, $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t$ and $\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t$ can involve n or less relation predicates $R(\bar{c}, _)$. We finalize our proof by discussing the remaining three cases.

Case $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t \equiv F$ derived from $Q_2 = Q_3 \bowtie Q_4$:

The assumption $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv F$ assures that at least one of the operands equals F. Without losing generality, we assume $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \equiv F$. Then, we conclude from the induction assumption (IA1) applied on $(\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle}$ that

$$\begin{aligned} (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \wedge (\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv F \wedge (\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \equiv F \end{aligned}$$

holds.

Case $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \vee \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t \equiv F$ derived from $Q_2 = Q_3 \cup Q_4$:

In this case, we can directly argue that $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t$ and $\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t$ must equal F. Otherwise, $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ could not be equal to F. Accordingly, by applying the induction assumption (IA1) for both operands, we obtain the following:

$$\begin{aligned} (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \vee \phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \vee (\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv F \vee F \equiv F. \end{aligned}$$

Case $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv \phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t) \equiv F$ derived from $Q_2 = Q_3 \setminus Q_4$:

Here, the first subcondition structure $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t$ must equal F and/or the second subcondition structure $\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t$ has to equal T. Otherwise, the resulting truth value T of $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t$ would contradict the assumption $\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t \equiv F$.

If we assume $\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \equiv F$, we exploit our induction assumption (IA1) and conclude the following:

$$\begin{aligned} (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t))_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \wedge (\neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t))_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv F \wedge (\neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t))_{\langle R(\bar{c}, _) | F \rangle} \equiv F. \end{aligned}$$

When $\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t \equiv T$ is given, we directly use our induction assumption (IA2):

$$\begin{aligned} (\phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t \wedge \neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t))_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \wedge (\neg(\phi_{Q_4, \langle \bar{x}_4 | d_{\bar{x}_4} \rangle}^t))_{\langle R(\bar{c}, _) | F \rangle} \\ &\equiv (\phi_{Q_3, \langle \bar{x}_3 | d_{\bar{x}_3} \rangle}^t)_{\langle R(\bar{c}, _) | F \rangle} \wedge \neg(T) \equiv F. \end{aligned}$$

□

14.3 Decomposed event relations

Throughout Chapter (13), we emphasized that our event relation \mathbf{E}_Q does not reflect any fixed formula structures for our final lineage formulas. Among other advantages, this gives us the freedom of generating \mathbf{E}_Q in different forms. We only have to ensure that all three necessary input parts, namely

$R_5(W^{max})$				$R_6(W^{max})$			
TID	A	B		TID	A	C	
t_{12}	1	2	e_{12}	t_{16}	1	6	e_{16}
t_{13}	1	3	e_{13}	t_{17}	1	7	e_{17}
t_{14}	1	4	e_{14}	t_{18}	1	8	e_{18}
t_{15}	1	5	e_{15}	t_{19}	2	9	e_{19}

Figure 14.1: Maximal set of possible tuples $R_5(W^{max})$ and $R_6(W^{max})$

- the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$,
- all relevant domains \mathbf{D}^t , and
- all involved atomic tuple events e_1, \dots, e_n, F ,

can be delivered.

In the following, we sketch out an alternative way of creating event relations. Instead of generating one large event relation \mathbf{E}_Q , our vertical construction algorithm also supports several smaller relations, see Section (16.2).

The decomposition of event relations can dramatically reduce the sizes and generation times of event relations within our relation database layer. It also leads to smaller input sizes for our probabilistic query engine. Both facts have a significant positive effect on our overall processing time.

To be more concrete, we tailor the well-known concept of *relation decomposition* in order to alternatively express an event relation \mathbf{E}_Q with a single *join relation* \mathbf{J}_Q and several *event source relations* $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$.

First of all, we introduce an example that can be used to compare event relations in composed and decomposed forms.

Example 14.1 (Example query for composed and decomposed event relation). *Let us consider following algebra query Q and its equivalent domain calculus query Q^c :*

$$Q = \pi_A(\sigma_{(B \neq 5) \wedge (B' \neq 2)}(R_5 \bowtie \rho_{(B' \leftarrow B)}(R_5))) \setminus \pi_A(R_6) \equiv \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t\} = Q^c$$

with the condition

$$\begin{aligned} \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle} &\equiv (\exists x_B, x'_B : \forall x_C : \phi^t(x_B, x'_B, x_C))_{\langle \bar{x} \mid \mathbf{D}^t \rangle} \\ &\equiv \bigoplus_{d_{x_B} \in \mathbf{D}_{x_B}^t} \left(\bigoplus_{d_{x'_B} \in \mathbf{D}_{x'_B}^t} \left(\bigoplus_{d_{x_C} \in \mathbf{D}_{x_C}^t} \phi_{\langle x_C \mid d_{x_C} \rangle}^t \right)_{\langle x'_B \mid d_{x'_B} \rangle} \right)_{\langle x_B \mid d_{x_B} \rangle} \end{aligned}$$

and the condition structure

$$\phi^t := (R_5(t_{x_A}, x_B) \wedge R_5(t_{x_A}, x'_B) \wedge ((x_B \neq 5) \wedge (x'_B \neq 2))) \wedge \neg R_6(t_{x_A}, x_C).$$

It is applied on a probabilistic database built from all possible tuple combinations of $R_5(W^{max})$ and $R_6(W^{max})$, see Figure (14.1) on Page (143).

For the sake of comparability, we first generate \mathbf{E}_Q using our original rules, which are presented in Definition (13.1) on Page (124).

\mathbf{E}_{R_5}				$\mathbf{E}_{\rho_{(B' \leftarrow B)}(R_5)}$			
	t_{x_A}	x_B	$R_5(t_{x_A}, x_B)$		t_{x_A}	x'_B	$R_5(t_{x_A}, x'_B)$
d_{12}	1	2	e_{12}	d_{16}	1	2	e_{12}
d_{13}	1	3	e_{13}	d_{17}	1	3	e_{13}
d_{14}	1	4	e_{14}	d_{18}	1	4	e_{14}
d_{15}	1	5	e_{15}	d_{19}	1	5	e_{15}

\mathbf{E}_{R_6}			
	t_{x_A}	x_C	$R_6(t_{x_A}, x_C)$
d_{20}	1	6	e_{16}
d_{21}	1	7	e_{17}
d_{22}	1	8	e_{18}
d_{23}	2	9	e_{19}

\mathbf{E}_Q							
	t_{x_A}	x_B	x'_B	x_C	$R_5(t_{x_A}, x_B)$	$R_5(x_A, x'_B)$	$R_6(t_{x_A}, x_C)$
d_{24}	1	2	3	6	e_{12}	e_{13}	e_{16}
d_{25}	1	2	3	7	e_{12}	e_{13}	e_{17}
d_{26}	1	2	3	8	e_{12}	e_{13}	e_{18}
d_{27}	1	2	4	6	e_{12}	e_{14}	e_{16}
d_{28}	1	2	4	7	e_{12}	e_{14}	e_{17}
d_{29}	1	2	4	8	e_{12}	e_{14}	e_{18}
d_{30}	1	2	5	6	e_{12}	e_{15}	e_{16}
d_{31}	1	2	5	7	e_{12}	e_{15}	e_{17}
d_{32}	1	2	5	8	e_{12}	e_{15}	e_{18}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
d_{48}	1	4	5	6	e_{14}	e_{15}	e_{16}
d_{49}	1	4	5	7	e_{14}	e_{15}	e_{17}
d_{50}	1	4	5	8	e_{14}	e_{15}	e_{18}

Figure 14.2: Composed event relation for $Q = \pi_A(\sigma_{(B \neq 5) \wedge (B' \neq 2)}(R_5 \bowtie \rho_{(B' \leftarrow B)}(R_5))) \setminus \pi_A(R_6)$

Example 14.2 (Composed event relation). \mathbf{E}_Q of Figure (14.2) on Page (144) contains 27 entries. This relatively high number of resulting domain values is caused by the two (m:n)-joins between the intermediate event relations

$$\mathbf{E}_{R_5} \quad \text{and} \quad \mathbf{E}_{\rho_{(B' \leftarrow B)}(R_5)}$$

as well as

$$\mathbf{E}_{\pi_A(\sigma_{((B \neq 5) \wedge (B' \neq 2))}(R_5 \bowtie \rho_{(B' \leftarrow B)}(R_5)))} \quad \text{and} \quad \mathbf{E}_{\pi_A(R_6)}.$$

This type of queries in particular can benefit from our alternative method.

Next, we explain our approach of event relation decomposition by describing:

- the structure of a special normal form Q^d inferred from the input query Q ,
- the construction of a join relation \mathbf{J}_Q , and
- the creation of our event source relations $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$.

Normal form Q^d

In the first step, we transform the given algebra query Q into a special normal form denoted as Q^d . When Q^d is considered in the form of a query tree, all projection and selection operations are pushed down in Q^d as much as possible. By doing so, we obtain a form where each relation operator is wrapped in a subquery of the form $\pi_A(\sigma_F(R))$.

When a relation is not assigned to a wrapping projection/selection operation, we simply insert an additional neutral projection/selection operation.

Example 14.3 (Normal form Q^d). *For our example query Q from Example (14.1) on Page (143), we obtain following equivalent normal form Q^d :*

$$Q \equiv (\pi_A(\sigma_{(B \neq 5)}(R_5)) \bowtie \pi_A(\sigma_{(B \neq 2)}(R_5))) \setminus \pi_A(\sigma_T(R_6)) = Q^d.$$

Join relation \mathbf{J}_Q

We can set up our joining relation \mathbf{J}_Q by using Q^d . In principle, it just comprises domain values that are necessary to compose event source relations \mathbf{S} containing all atomic tuple events. Then, the joining relation \mathbf{J}_Q does not need to store any atomic tuple events. Apart from this splitting, we basically reuse all rules known from Definition (13.1) on Page (124).

Definition 14.1 (Construction of join relation \mathbf{J}_{Q^d}). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D} \rangle}^t\} \equiv \{t \mid \overline{\bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} \mid d \rangle}^t}\}.$$

If Q^d is its normal form for computing a relevant domain in a decomposed form, we can recursively generate the join relation \mathbf{J}_{Q^d} as follows:

$$\begin{aligned} Q^d = \pi_A(\sigma_F(R)) \text{ with } R(\bar{x}, t_{\bar{y}}) \text{ in } \phi^t & : \mathbf{J}_{Q^d} := \rho_{((\bar{x}, \bar{y}) \leftarrow \text{vars}^{-1}(\bar{x}, \bar{y}))}(\sigma_F(R(W^{max}))) \\ Q^d = \sigma_F(Q_1^d), Q_1^d \neq R & : \mathbf{J}_{Q^d} := \sigma_F(\mathbf{J}_{Q_1^d}) \\ Q^d = \pi_A(Q_1^d), Q_1^d \neq \sigma_F(R) & : \mathbf{J}_{Q^d} := \mathbf{J}_{Q_1^d} \\ Q^d = Q_1^d \bowtie Q_2^d & : \mathbf{J}_{Q^d} := \mathbf{J}_{Q_1^d} \bowtie \mathbf{J}_{Q_2^d} \\ Q^d = Q_1^d \cup Q_2^d & : \mathbf{J}_{Q^d} := (\mathbf{J}_{Q_1^d} \times \{(_)_{\bar{h}_1}\}) \cup (\{(_)_{\bar{h}_2}\} \times \mathbf{J}_{Q_2^d}) \\ Q^d = Q_1^d \setminus Q_2^d & : \mathbf{J}_{Q^d} := \mathcal{M}_1 \dot{\cup} \mathcal{M}_2, \\ & \mathcal{M}_1 := \mathbf{J}_{Q_1^d} \bowtie \mathbf{J}_{Q_2^d} \\ & \mathcal{M}_2 := (\mathbf{J}_{Q_1^d} \setminus \pi_{\text{attr}(\mathbf{J}_{Q_1^d})}(\mathcal{M}_1)) \times \{(_)_{\bar{h}_1}\} \\ Q^d = \rho_{(B \leftarrow A)}(Q_1^d) & : \mathbf{J}_{Q^d} := \rho_{(\text{vars}(B) \leftarrow \text{vars}(A))}(\mathbf{J}_{Q_1^d}). \end{aligned}$$

Thereby,

- in the relation rule $Q^d = R$, we label the attribute storing e_i with a corresponding relation predicate $R(\bar{x}, t_{\bar{y}})$ of ϕ^t and
- in the union and difference rule $Q^d = Q_1^d \cup Q_2^d$ and $Q^d = Q_1^d \setminus Q_2^d$, we use our symbol $_$ introduced in Definition (11.4) on Page (104) in order to fill the missing attribute columns:

$$\bar{h}_1 := (\text{attr}(\mathbf{J}_{Q_2^d}) \setminus \text{attr}(\mathbf{J}_{Q_1^d})) \quad \text{and} \quad \bar{h}_2 := (\text{attr}(\mathbf{J}_{Q_1^d}) \setminus \text{attr}(\mathbf{J}_{Q_2^d})).$$

Example 14.4 (Join relation \mathbf{J}_{Q^d}). *We show in Figure (14.3) on Page (146) the join relation \mathbf{J}_Q created for our query of Example (14.3) on Page (145). It only involves one domain value $(1)_{t_{x_A}}$ necessary to compose the event source relations $\mathbf{S}_{R_5(t_{x_A}, x_B)}$, $\mathbf{S}_{R_5(t_{x_A}, x'_B)}$, and $\mathbf{S}_{R_6(t_{x_A}, x_C)}$ to \mathbf{E}_Q .*

<table><tr><th>\mathbf{J}_Q</th></tr><tr><td>t_{x_A}</td></tr><tr><td>1</td></tr></table>	\mathbf{J}_Q	t_{x_A}	1	<table><tr><th colspan="3">$\mathbf{S}_{R_5(t_{x_A}, x_B)}$</th></tr><tr><th>$t_{x_A}$</th><th>$x_B$</th><th>$R_5(t_{x_A}, x_B)$</th></tr><tr><td>1</td><td>2</td><td>e_{12}</td></tr><tr><td>1</td><td>3</td><td>e_{13}</td></tr><tr><td>1</td><td>4</td><td>e_{14}</td></tr></table>			$\mathbf{S}_{R_5(t_{x_A}, x_B)}$			t_{x_A}	x_B	$R_5(t_{x_A}, x_B)$	1	2	e_{12}	1	3	e_{13}	1	4	e_{14}
\mathbf{J}_Q																					
t_{x_A}																					
1																					
$\mathbf{S}_{R_5(t_{x_A}, x_B)}$																					
t_{x_A}	x_B	$R_5(t_{x_A}, x_B)$																			
1	2	e_{12}																			
1	3	e_{13}																			
1	4	e_{14}																			

<table><tr><th colspan="3">$\mathbf{S}'_{R_5(t_{x_A}, x'_B)}$</th></tr><tr><th>$t_{x_A}$</th><th>$x'_B$</th><th>$R_5(t_{x_A}, x'_B)$</th></tr><tr><td>1</td><td>3</td><td>e_{13}</td></tr><tr><td>1</td><td>4</td><td>e_{14}</td></tr><tr><td>1</td><td>5</td><td>e_{15}</td></tr></table>			$\mathbf{S}'_{R_5(t_{x_A}, x'_B)}$			t_{x_A}	x'_B	$R_5(t_{x_A}, x'_B)$	1	3	e_{13}	1	4	e_{14}	1	5	e_{15}	<table><tr><th colspan="3">$\mathbf{S}_{R_6(t_{x_A}, x_C)}$</th></tr><tr><th>$t_{x_A}$</th><th>$x_C$</th><th>$R_6(t_{x_A}, x_C)$</th></tr><tr><td>1</td><td>6</td><td>e_{16}</td></tr><tr><td>1</td><td>7</td><td>e_{17}</td></tr><tr><td>1</td><td>8</td><td>e_{18}</td></tr></table>			$\mathbf{S}_{R_6(t_{x_A}, x_C)}$			t_{x_A}	x_C	$R_6(t_{x_A}, x_C)$	1	6	e_{16}	1	7	e_{17}	1	8	e_{18}
$\mathbf{S}'_{R_5(t_{x_A}, x'_B)}$																																			
t_{x_A}	x'_B	$R_5(t_{x_A}, x'_B)$																																	
1	3	e_{13}																																	
1	4	e_{14}																																	
1	5	e_{15}																																	
$\mathbf{S}_{R_6(t_{x_A}, x_C)}$																																			
t_{x_A}	x_C	$R_6(t_{x_A}, x_C)$																																	
1	6	e_{16}																																	
1	7	e_{17}																																	
1	8	e_{18}																																	

Figure 14.3: Decomposed event relation $\mathbf{E}_Q = \mathbf{J}_Q \bowtie \mathbf{S}_{R_5(t_{x_A}, x_B)} \bowtie \mathbf{S}'_{R_5(t_{x_A}, x'_B)} \bowtie \mathbf{S}_{R_6(t_{x_A}, x_C)}$.

Event source relations

In order to set up the remaining event source relations $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$, we basically apply all subqueries of the form $\pi_A(\sigma_F(R))$ on the respective maximal possible sets of tuples.

Definition 14.2 (Construction of event source relation $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \bigwedge \phi_{\langle \bar{x} \mid d \rangle}^t\}.$$

If Q^d is the normal form for generating a decomposed event relation, we build for each relation predicate $R(\bar{x}, \bar{c})$ of ϕ^t an event source relation $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$:

$$\mathbf{S}_{R(\bar{x}, t_{\bar{y}})} := \rho_{((\bar{x}, \bar{y}) \leftarrow \text{vars}^{-1}(\bar{x}, \bar{y}))}(\{(t, e_t) \mid t \in \pi_A(\sigma_F(R(W^{max})))\}),$$

where

- $\pi_A(\sigma_F(R))$ is the wrapping subquery of R in Q^d and
- $R(\bar{x}, t_{\bar{y}})$ is the corresponding relation predicate in ϕ^t .

Example 14.5 (Event source relations). *Our final event source relations*

$$\begin{aligned} \mathbf{S}_{R_5(t_{x_A}, x_B)} &= \rho_{((t_{x_A}, x_B) \leftarrow \bar{A}, B)}(\{(t, e_t) \mid t \in \pi_{\bar{A}, B}(\sigma_{(B \neq 5)}(R_5(W^{max})))\}), \\ \mathbf{S}_{R_5(t_{x_A}, x'_B)} &= \rho_{((t_{x_A}, x'_B) \leftarrow \bar{A}, B)}(\{(t, e_t) \mid t \in \pi_{\bar{A}, B}(\sigma_{(B \neq 2)}(R_5(W^{max})))\}), \quad \text{and} \\ \mathbf{S}_{R_6(t_{x_A}, x_C)} &= \rho_{((t_{x_A}, x_C) \leftarrow \bar{A}, C)}(\{(t, e_t) \mid t \in \pi_{\bar{A}, C}(R_6(W^{max})))\} \end{aligned}$$

are shown in Figure (14.3). They carry all required atomic tuple events.

In conjunction with \mathbf{J}_Q of Example (14.4) on Page (145), all event source relations can be combined as follows:

$$\mathbf{E}_Q = \mathbf{J}_Q \bowtie \mathbf{S}_{R_5(t_{x_A}, x_B)} \bowtie \mathbf{S}_{R_5(t_{x_A}, x'_B)} \bowtie \mathbf{S}_{R_6(t_{x_A}, x_C)}.$$

This operation is indirectly executed within our adjusted **vlc**-algorithm shown in Section (16.2). In contrast to 27 domain values of Example (14.2), our algorithm only need to process 10 input tuples in its adjusted version.

Chapter 15

Vertical lineage construction

In this chapter, we present our central vertical lineage construction algorithm, which is performed within our probabilistic query engine. The first two sections of this chapter cover the following themes:

- Section (15.1): the basic ideas of our vertical construction algorithm and
- Section (15.2): the pragmatic benefits of our method experimentally.

Please note that we focus on the construction aspects of our algorithm in this chapter. Its capability of manipulating the overall lineage structures independently from the relevant domains is extensively studied in the next chapter.

First of all, we recap the requirements for our algorithm already outlined in Chapter (8). Figure (15.1) on Page (148) shows the corresponding satisfaction matrix of the stated design goals, which are derived from the advantages and disadvantages of the most important state-of-the-art approaches for lineage construction.

15.1 Vertical lineage construction algorithm

The main ideas of our **vlc**-algorithm were already compactly presented in Chapter (9). In this overview, we simplified some aspects, which are now explained more deeply:

- vertical construction principle,
- primary data structures,
- iterative main loop, and
- recursive paths insertion.

We start our discourse with an overview of the two main inputs for our algorithm:

- Algorithm input (1): event relation \mathbf{E}_Q : We already explained in Chapter (13) how to efficiently generate an event relation \mathbf{E}_Q . Most notably, an event relation contains all possible answers, all relevant domains and all required atomic tuple events for constructing the desired lineage formulas.
- Algorithm input (2): input query Q : The algebra query Q *indirectly* determines the overall structure of all lineage formulas via its inferred condition structure ϕ^t . Since we focus on lineage formulas built for an algebra query Q , we can define the instructions of our main algorithm directly over the structure of Q .

Example 15.1 (Main inputs for vertical lineage construction algorithm). *Figure (15.2) and (15.3) on Page (148) and (149) depict the two main inputs for constructing the lineage formulas of Q_∞ of Example (5.1) on Page (30) with respect to the sample database of Example (4.1) on Page (24):*

Fulfillment of design goals for lineage construction					
	design goal	safe plans (MystiQ)	nested (SPROUT2)	DNF (MayBMS)	networks (PrDB/Trio)
1	full relational algebra support	no	yes	no	yes
2	unlimited lineage formula lengths	-	no	yes	yes
3	native relational operator and data types	yes	no	yes	yes
4	result set sizes as deterministic case	yes	yes	yes	no

Figure 15.1: Fulfillment of design goals for lineage construction

$\mathbf{E}_{Q_{\bowtie}}$					
	t_{x_A}	x_B	$R_1(x_A)$	$R_2(x_A, x_B)$	$R_3(x_A, x_B)$
d_1	1	3	e_1	e_3	e_5
d_2	1	4	e_1	e_4	e_6

Figure 15.2: Algorithm input (1): event relation $\mathbf{E}_{Q_{\bowtie}}$

- the event relation $\mathbf{E}_{Q_{\bowtie}}$ containing all possible answers, relevant domain values and required atomic tuple events, and
- the query Q_{\bowtie} implicitly providing the mapping between algebra operators, logical operators, and atomic tuple events as defined by our conceptional transformation chain of Chapter (10).

Vertical construction principle

The classical lineage construction rules by Fuhr and Röllecke (Lemma (6.1) on Page (35)) propose a recursive combination of subformulas. It starts with a set of atomic tuple events and ends up with a complex lineage formula. When lineage formulas are represented as formula trees, the classical rules create tree structures which mainly grow *horizontally*. In each construction step, they connect smaller formula trees at their root nodes in order to obtain a larger formula tree.

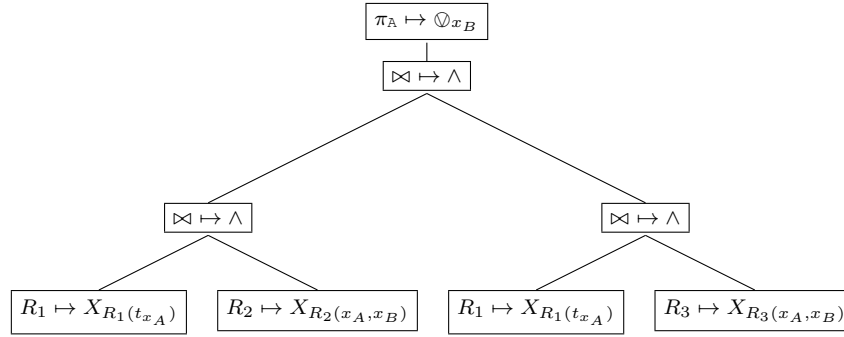
Our construction principle is different. Instead of combining already built subtrees, we insert *complete* tree paths *vertically* into our emerging formula tree.

The vertical construction principle coalesces the provided atomic tuple events with the desired lineage structures.

For this purpose, we always navigate from the root node to a set of leaf nodes. When there are no nodes on a visited path, we create them instantly on our way down. Thereby, the control flow of our algorithm as well as the navigation through the evolving formula tree are controlled by the implicitly given mapping between algebra operators, logical operators, and atomic tuple events. That means, our algorithm internally implements the MAC rules of Lemma (10.1) on Page (84), which is applied on the input query Q .

Despite of exploiting all theoretical concepts presented in Part (IV), we can keep our algorithm relatively simple. It only encompasses two parts that can be both implemented very easily.

The first function $\mathbf{vlc}(t, \mathbf{E}_Q, Q)$ (Algorithm (3) on Page (149)) contains the main loop that primarily iterates over all relevant domain values d with their associated atomic tuple events $d.e$. Each atomic tuple event $d.e$ initiates the insertion of a set of tree paths, which have $d.e$ as leaf. For

Figure 15.3: Algorithm input (2): implicit transformation mapping based on Q_{\bowtie} **Algorithm 3:** $\text{vlc}(t, \mathbf{E}_Q, Q)$

```

1  $rootNode := \boxed{F}$ ;
2 foreach  $d \in \mathbf{E}_Q$  do
3   if  $(d_{\text{vars}(\text{head}(Q))} = t)$  then
4     foreach  $d.e \in (\text{atomicTupleEvents}(d) \setminus \{d.F\})$  do
5        $\text{insertPaths}(d.e, Q, rootNode)$ ;
6     end
7   end
8 end
9 return  $rootNode$ ;

```

this purpose, the main loop calls our second main function $\text{insertPaths}(d.e, Q, node)$ shown in Algorithm (4) on Page (150). It adds tree paths to the evolving formula tree in a vertical fashion. Thereby, all paths are traversed from the tree root to the leaves node-by-node. Before we describe both main functions in more detail, we have a closer look at the primary components our formula trees consist of.

Primary data structures

Each node of a formula tree has one of the following types:

$$node \in \{\boxed{\vee}, \boxed{\wedge}, \boxed{\bigvee}, \boxed{\wedge\neg}, \boxed{\text{d.e}}, \boxed{F}\}.$$

All nodes are connected through pointers heading from the tree root to the leaves. If a specific node represents a binary operator (i.e., $\boxed{\wedge}$, $\boxed{\vee}$, or $\boxed{\wedge\neg}$) that node has two pointers $node.leftChild$ and $node.rightChild$, which refers to its left and right child node, respectively.

On the other hand, an n-ary disjunction node $\boxed{\bigvee}$ implements the pointers to its child nodes by means of a map data structure denoted as $node.children$. We use domain values of the attributes as keys for $node.children$, that are projected out by the corresponding algebra projection. We will describe this mechanism below.

Iterative main loop $\text{vlc}(t, \mathbf{E}_Q, Q)$

The final outcome of our main function $\text{vlc}(t, \mathbf{E}_Q, Q)$ is a lineage formula for a specific tuple t .

Algorithm 4: $\text{insertPaths}(d.e, Q, \text{node})$

```

1 switch  $Q$  do
2   case  $Q = \pi_{\mathcal{A}}(Q_1)$ 
3      $\text{node} := \boxed{\mathbb{V}}$ ;
4      $\text{key} := d_{\text{vars}(\text{head}(Q_1) \setminus \mathcal{A})}$ ;
5      $\text{insertPaths}(d.e, Q_1, \text{createIfAbsent}(\text{node.children}[\text{key}]))$ ;
6   case  $Q = Q_1 \Theta Q_2$  with  $\Theta \in \{\bowtie, \cup, \setminus\}$ 
7      $\text{node} := \begin{cases} \boxed{\wedge} & \text{if } \Theta = \bowtie \\ \boxed{\vee} & \text{if } \Theta = \cup \\ \boxed{\wedge \neg} & \text{if } \Theta = \setminus \end{cases}$ 
8     if  $((\text{relPredicatesMappedFrom}(d.e) \cap \text{relPredicates}(\phi_{Q_1}^t)) \neq \emptyset)$  then
9        $\text{insertPaths}(d.e, Q_1, \text{createIfAbsent}(\text{node.leftChild}))$ ;
10      if  $(\text{absent}(\text{node.rightChild}))$  then
11         $\text{node.rightChild} := \boxed{\text{F}}$ ;
12      end
13    end
14    if  $((\text{relPredicatesMappedFrom}(d.e) \cap \text{relPredicates}(\phi_{Q_2}^t)) \neq \emptyset)$  then
15       $\text{insertPaths}(d.e, Q_2, \text{createIfAbsent}(\text{node.rightChild}))$ ;
16      if  $(\text{absent}(\text{node.leftChild}))$  then
17         $\text{node.leftChild} := \boxed{\text{F}}$ ;
18      end
19    end
20  case  $Q = \sigma_F(Q_1)$ 
21    if  $(F(d) = T)$  then
22       $\text{insertPaths}(d.e, Q_1, \text{node})$ ;
23    else
24       $\text{node} := \boxed{\text{F}}$ ;
25    end
26  case  $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$ 
27     $\text{insertPaths}(d.e, Q_1, \text{node})$ ;
28  case  $Q = R$ 
29     $\text{node} := \boxed{d.e}$ ;
30  endsw
31 endsw

```

We obtain all lineage formulas by computing

$$\forall t \in \pi_{\text{vars}(\text{head}(Q))}(\mathbf{E}_Q) : \varphi_t := \mathbf{vlc}(t, \mathbf{E}_Q, Q),$$

since our event input relation \mathbf{E}_Q includes all possible answers, see Lemma (14.1) on Page (137).

The iterative main loop of Algorithm (3) on Page (149) represents the essential part of $\mathbf{vlc}(t, \mathbf{E}_Q, Q)$. It initially selects all domain values that belong to a specific relevant domain \mathbf{D}^t , see Line (2) and (3) in Algorithm (3) on Page (149) and Lemma (13.3) on Page (125).

By means of a further inner loop, we iterate over all atomic tuple events $d.e$ provided for the current domain value d , see Line (4). All needed atomic tuple events are delivered through the auxiliary function $\mathbf{atomicTupleEvents}(d)$. To preserve the connection between the domain value d and one of its atomic tuple events e , we use the notation $d.e$.

Example 15.2 (Auxiliary function **atomicTupleEvents**(d)). If we identify the atomic tuple events for our domain values d_1 and d_3 given in $\mathbf{E}_{Q_{\bowtie}}$ and $\mathbf{E}_{Q_{\cup}}$ (Figure (15.2) and (15.5)), our auxiliary function provides

$$\begin{aligned} \mathbf{atomicTupleEvents}(d_1) &= \{d_1.e_1, d_1.e_3, d_1.e_5\} \quad \text{and} \\ \mathbf{atomicTupleEvents}(d_3) &= \{d_3.e_4, d_3.e_1, d_3.F\}. \end{aligned}$$

In general, every atomic tuple event $d.e$ can form the leaves of several tree paths. The respective insertion processes are initiated in Line (5) by calling our second main function **insertPaths**(...). In contrast to the introductory version of our algorithm presented in Chapter (9), we now omit the explicit creation of paths with the truth value $d.F$ as leaf. That means, all nodes of type \boxed{F} within the evolving formula tree are *indirectly* inserted in Line (11), (17), or (24) of Algorithm (4). The underlying optimization techniques are explained and justified as advanced topics in Chapter (16).

Recursive insertion of paths with **insertPaths**($d.e, Q, node$):

The main job of function **insertPaths**($d.e, Q, node$) is the creation of paths within our emerging formula tree. To perform this task, the control flow of our algorithm moves path-wise through the given input Q and the evolving formula tree in parallel. In this sense, the parameter Q of **insertPaths**($d.e, Q, node$) represents the current position in the input query, while the parameter $node$ gives the corresponding position on the traversed path of the emerging formula tree. This means that our algorithm *recursively* navigates from the tree root to the respective leaves *node-by-node* for each atomic tuple event $d.e$.

Implementing conceptional transformation chain

A non-existing node has to be created whenever our algorithm intends to visit it for the first time, see Line (5), (9), and (15) of Algorithm (4) on Page (150). The assignment of node types are based on the top-most operator of the current algebra query Q . Most importantly, in Line (3), (7), and (29), we apply our conceptional mapping between algebra operators, logical operators and atomic tuple events known from Chapter (10), e.g.,

$$\begin{array}{ccccccc} \underbrace{Q = \pi_A(Q_1)}_{\text{relational algebra}} & \mapsto & \underbrace{(\bigvee \phi_{Q_1})}_{\text{relevant condition}} & \mapsto & \underbrace{(\bigvee \varphi_{Q_1})}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{\bigvee}}_{\text{tree node}} \\ \underbrace{Q = Q_1 \bowtie Q_2}_{\text{relational algebra}} & \mapsto & \underbrace{(\phi_{Q_1} \wedge \phi_{Q_2})}_{\text{relevant condition}} & \mapsto & \underbrace{(\varphi_{Q_1} \wedge \varphi_{Q_2})}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{\wedge}}_{\text{tree node}} \\ \underbrace{Q = R}_{\text{relational algebra}} & \mapsto & \underbrace{R(\bar{x})}_{\text{relevant condition}} & \mapsto & \underbrace{X_{R(\bar{x})}}_{\text{lineage formula}} & \mapsto & \underbrace{\boxed{d.e}}_{\text{tree node}}. \end{array}$$

Our algorithm directly construct tree nodes based on algebra operators. The intermediate mappings involving relevant conditions are not materialized practically. We do not have to perform any intermediate transformations into PNF.

Navigating through the input query and the emerging formula tree

In Line (5), (9), and (15), our algorithm navigates by the following means:

- the key values of $node.children$ used in the case $Q = \pi_A(Q_1)$ and

- the two supporting functions **relPredicatesMappedFrom** and **relPredicates** applied in the cases $Q = Q_1 \Theta Q_2$ with $\Theta \in \{\bowtie, \cup, \setminus\}$.

Navigation choice in the case $Q = \pi_{\mathcal{A}}(Q_1)$: The first type of junction occurs, if our algorithm maps a projection operation $\pi_{\mathcal{A}}(Q_1)$ to an n-ary disjunction node $\boxed{\vee}$. Then, the next node to visit is picked by a key for *node.children*, which is extracted from the current domain value d , see Line (4).

Our idea for computing that key is to utilize the variables projected out by the corresponding projection operation $\pi_{\mathcal{A}}(Q_1)$. The variable values of $\text{vars}(\text{head}(Q_1) \setminus \mathcal{A})$ can be exploited as proper keys, since they are always unique for a specific \vee -operand.

Example 15.3 (Pointer map for an n-ary operation node). *We exemplify the handling of an n-ary operation by considering the top-most projection of our example query:*

$$Q_{\bowtie} \equiv \pi_{\mathcal{A}}((R_1 \bowtie R_2) \bowtie (R_1 \bowtie R_3)).$$

In Figure (15.3) on Page (149), we already depicted our implicitly given mapping based on Q_{\bowtie} . We also know from Example (13.4) on Page (128) that the relevant domain for Q_{\bowtie} is determined to $\mathbf{D}^t = \{(3)_{x_B}, (4)_{x_B}\}$. So, the derived relevant condition $\Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^t \rangle}^t$ combines the substituted condition structures $\phi_{\bowtie, \langle x_B | 3 \rangle}^t$ and $\phi_{\bowtie, \langle x_B | 4 \rangle}^t$ disjunctively. Thereby, both subformulas can be uniquely identified with their corresponding substituting domain values $(3)_{x_B}$ and $(4)_{x_B}$:

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^t \rangle}^{(1)} &= \phi_{\bowtie, \langle x_B | 3 \rangle}^{(1)} \vee \phi_{\bowtie, \langle x_B | 4 \rangle}^{(1)} \\ &\equiv \underbrace{((R_1(1) \wedge R_2(1, 3)) \wedge (R_1(1) \wedge R_3(1, 3)))}_{\text{uniquely identified by } (3)_{x_B}} \vee \\ &\quad \underbrace{((R_1(1) \wedge R_2(1, 4)) \wedge (R_1(1) \wedge R_3(1, 4)))}_{\text{uniquely identified by } (4)_{x_B}} \\ &\mapsto \underbrace{((e_1 \wedge e_3) \wedge (e_1 \wedge e_5))}_{\text{uniquely identified by } (3)_{x_B}} \vee \underbrace{((e_1 \wedge e_4) \wedge (e_1 \wedge e_6))}_{\text{uniquely identified by } (4)_{x_B}} \\ &= \varphi_{\bowtie}^t. \end{aligned}$$

We exploit this property for organizing the pointer map *node.children* of the top-most n-ary disjunction operation of Q_{\bowtie} by using the unique domain values $(3)_{x_B}$ and $(4)_{x_B}$ as keys. Accordingly, in our constructed formula tree, the keys $(3)_{x_B}$ and $(4)_{x_B}$ uniquely point to the subtrees containing the subformulas $\phi_{\bowtie, \langle x_B | 3 \rangle}^t$ and $\phi_{\bowtie, \langle x_B | 4 \rangle}^t$, see Figure (15.4) on Page (154).

Each key value required for *node.children* can be extracted from the current domain value d (Line (5) in Algorithm (4) on Page (150)), if we just consider the variables for the attributes that are projected out by the corresponding projection operation. In our case, Q_{\bowtie} indirectly projects its output tuples onto the variable x_A . Consequently, the values of variables x_B are projected out after its last operation.

Obviously, we can apply our concept of generating keys for *node.children* on each algebra subquery, that has a projection operation as top-most operation. In fact, projection operations do not have to be the last operations in an overall input query. In these cases, we just deal with the respective subconditions and subdomains.

Navigation choice in the cases $Q = Q_1 \Theta Q_2$ with $\Theta \in \{\bowtie, \cup, \setminus\}$: Here, the two auxiliary functions

$$\text{relPredicatesMappedFrom}(d.e) \quad \text{and} \quad \text{relPredicates}(\phi^t)$$

support our algorithm to decide where to move next.

The first function returns relation predicates, where the considered atomic tuple event $d.e$ is conceptionally mapped from. Our second supporter function determines all relation predicates of the current underlying condition structure.

Example 15.4 (Auxiliary functions). *Following relational predicates are returned for Q_{\bowtie} :*

$$\begin{aligned} \text{relPredicatesMappedFrom}(d_1.e_1) &= \{R_1(x_A)\} \\ \text{relPredicatesMappedFrom}(d_1.e_3) &= \{R_2(x_A, x_B)\} \\ \text{relPredicates}(\phi_{(R_1 \bowtie R_3)}^t) &= \{R_1(x_A), R_3(x_A, x_B)\} \\ \text{relPredicates}(\phi_{Q_{\bowtie}}^t) &= \{R_1(x_A), R_2(x_A, x_B), R_3(x_A, x_B)\}. \end{aligned}$$

These information can be easily taken from the column labels of $\mathbf{E}_{Q_{\bowtie}}$, and the implicitly given transformation mapping based on Q_{\bowtie} , see Example (15.1) on Page (147).

By intersecting both function outcomes, our algorithm checks whether the atomic tuple event $d.e$ is mapped from a relation predicate of the current underlying condition structure ϕ^t , see Line (8) and (14).

As described earlier, a condition structure ϕ^t describes the logical combination of all relation predicates. Therefore, it indirectly determines the logical combination of all atomic tuple events. Thus, in each recursive step, we narrow down the respective condition structure until the correct tree positions of an atomic tuple event $d.e$ are reached. Thereby, an already inserted atomic tuple event $d_i.e$ can be overwritten by another atomic tuple event $d_j.e$, e.g., $d_5.e_1$ by $d_6.e_1$ in Figure (15.6).

Theorem 15.1 (Vertical lineage construction algorithm). *Let Q be an algebra query with its event relation \mathbf{E}_Q . When $\text{vlc}(t, \mathbf{E}_Q, Q)$ constructs a lineage formula for a given tuple t , i.e., $\varphi^t := \text{vlc}(t, \mathbf{E}_Q, Q)$, we can determine the answer probability of t as*

$$\mathbf{P}(t \in Q) = \mathbf{P}(\varphi^t).$$

Proof. See Section (16.1). □

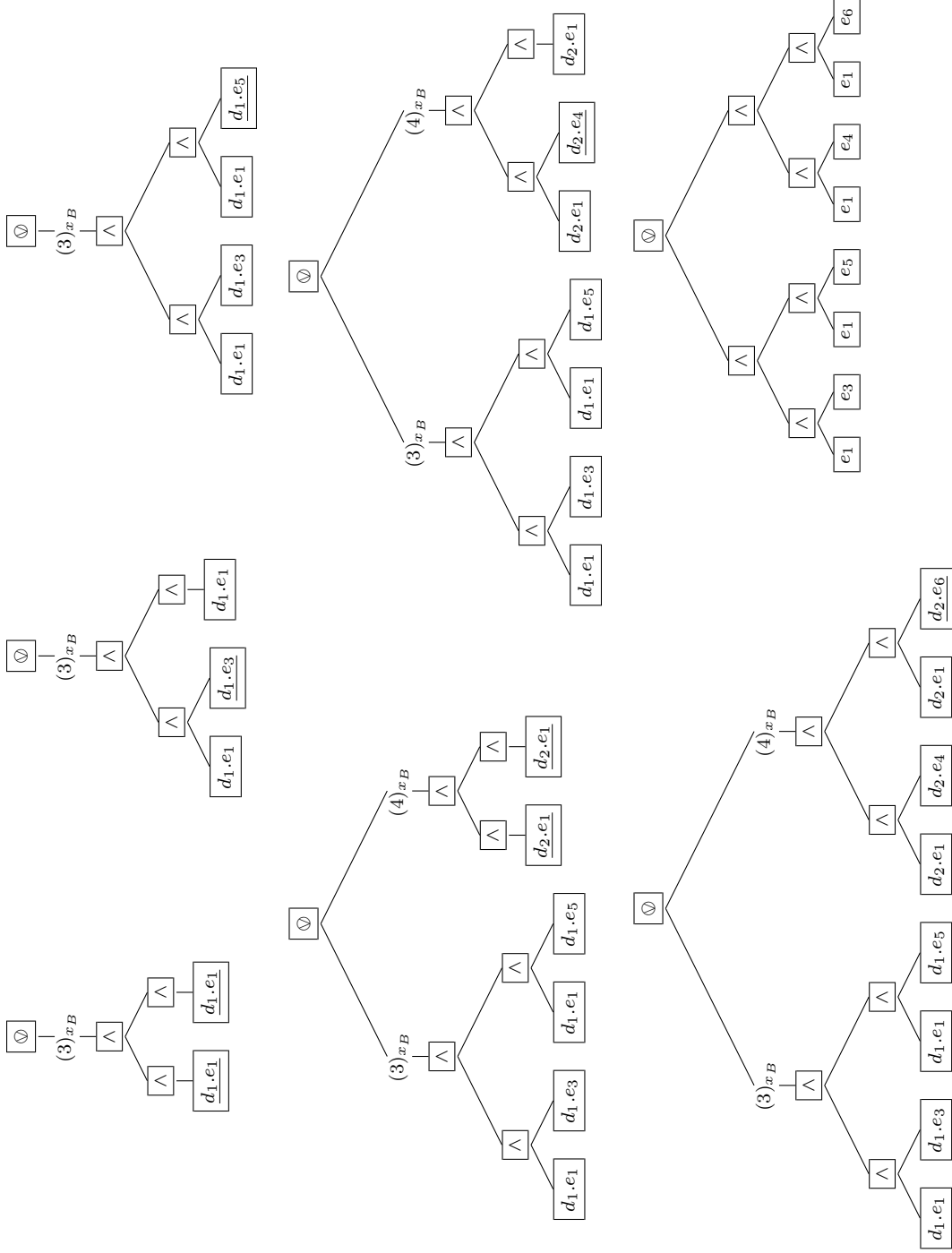
Example 15.5 (Vertical lineage construction). *In Figure (15.3) on Page (149), we illustrate the implicitly given mapping for Q_{\bowtie} . It provides the base for constructing the formula tree for our lineage formula*

$$\varphi_{\bowtie}^{(1)} := \text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\bowtie}}, Q_{\bowtie}).$$

The path-wise creation of $\varphi_{\bowtie}^{(1)}$ is depicted in Figure (15.4) on Page (154). The different formula trees show the intermediate construction results after inserting the tree paths initiated by $d_1.e_1, d_1.e_3, d_1.e_5, d_2.e_1, d_2.e_4$ and $d_2.e_6$.

In addition to the intermediate results, we also give the formula tree of $\varphi_{\bowtie}^{(1)}$ without any additional information.

In general, we always produce lineage formulas that are logically equivalent to the classical lineage formulas of Lemma (6.1) on Page (35), see Theorem (16.1) on Page (169). But they are not necessarily syntactically identical. To give examples, we refer to our formula trees built for the lineage formulas of Q_{\cup} and Q_{\setminus} of Example (5.1) on Page (30) with respect to the sample database of Example (4.1) on Page (24). They are illustrated in Figure (15.5) and (15.6) on Page (155) and (156).

Figure 15.4: Evolving formula tree for Q_{\bowtie} built by the following order of inserted atomic tuple events: $d_1.e_1, d_1.e_3, d_1.e_5, d_2.e_1, d_2.e_4, d_2.e_6$.

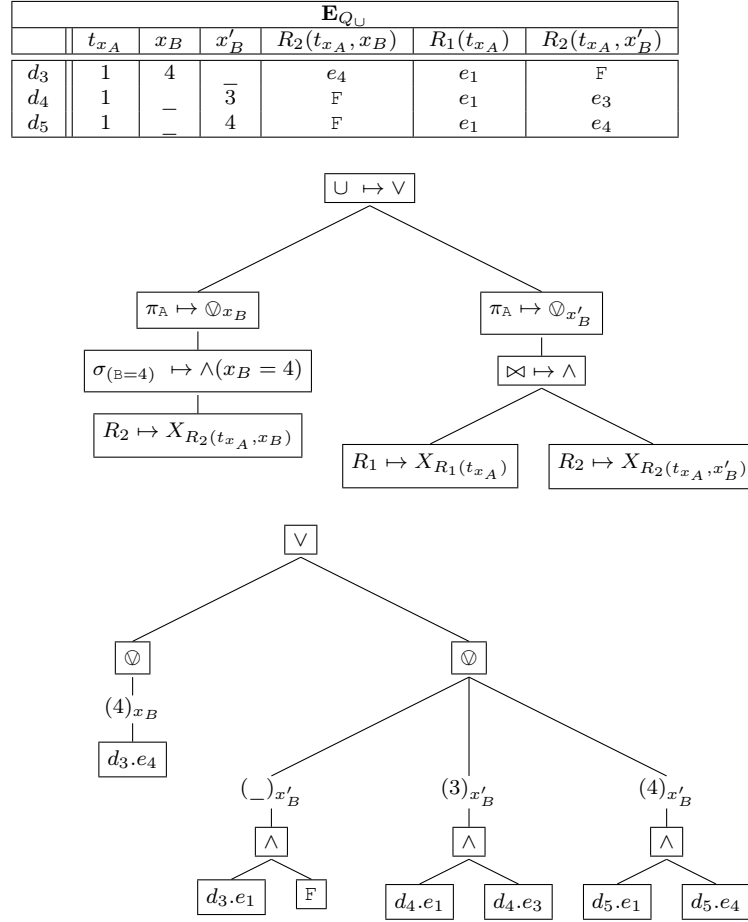


Figure 15.5: Event relation \mathbf{E}_{Q_U} , the implicitly given transformation mapping based on Q_U and the constructed formula tree of $\varphi_{Q_U}^{(1)}$

Lemma 15.1 (Complexity of vertical lineage construction algorithms). *Let Q be an algebra query, which is applied on a probabilistic database \mathbf{pdb} built from all possible tuple combinations of $R_1(W^{max}), \dots, R_n(W^{max})$. Then, the number of steps needed for determining the query result $Q(\mathbf{pdb})$ is bounded by*

$$|Q|^2 * \max(|R_1(W^{max})|, \dots, |R_n(W^{max})|)^{|Q|},$$

if we use our vertical lineage construction algorithm laid out in Algorithm (3) and (4) on Page (149) and (150).

Proof. See Section (16.1). □

15.2 Experiments: lineage construction

In this section, we discuss the results of our experiments, which were carried out to study the lineage construction part within a probabilistic query engine. Please remember that we already described in Section (13.5) our experiments, which were used to explore the relational database

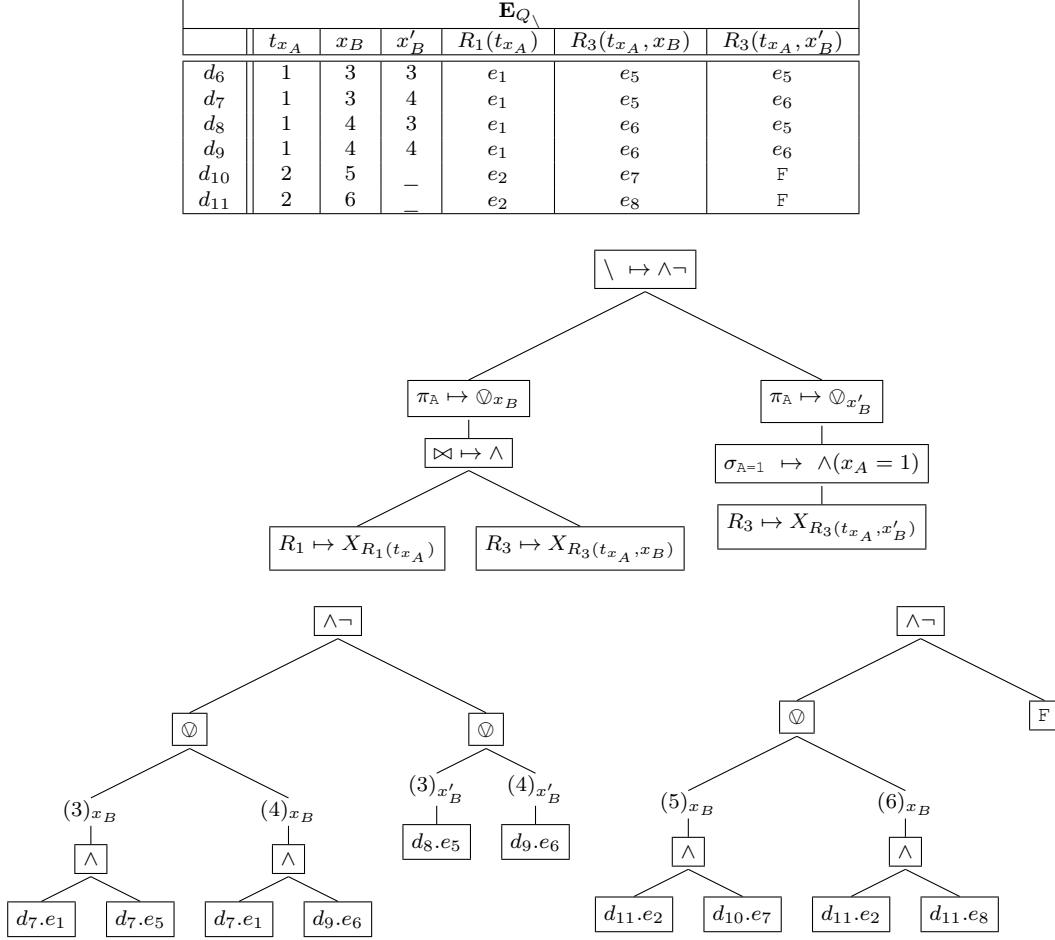


Figure 15.6: Event relation $\mathbf{E}_{Q_{\setminus}}$, implicitly given transformation mapping based on Q_{\setminus} and constructed formula trees of $\varphi_{Q_{\setminus}}^{(1)}$ and $\varphi_{Q_{\setminus}}^{(2)}$

layer. The produced relations gave us the inputs for the secondary-storage algorithms of our five basic approaches introduced in Section (13.5). We present the following topics in more detail:

- important implementation notes for our five basic approaches in terms of lineage construction,
- the data we measured in the experiments, and
- our experimental observations.

Lineage construction: safe plans

We mentioned earlier that the safe plans of [94] are only carried out within the relational database layer. As a consequence, there is no lineage construction within a probabilistic query engine.

Lineage construction: lineage formulas in DNF

The well-structured form of a lineage formulas in DNF can be easily created and managed as a list of conjuncts where a single conjunct is stored as sequence of atomic tuple events.

Analog to our first series of experiments, we had to omit the queries IMDB-Q4, IMDB-Q5, IMDB-Q6, TPC-H-Q5, TPC-H-Q6, and TPC-H-Q7, since they are not from **SPJ**.

```

{
  "binary_or": [
    {
      "n_ary_or": [
        {
          "atomic_tuple_event": {
            "id": "1",
            "prob": "0.3"
          }
        },
        {
          "atomic_tuple_event": {
            "id": "2",
            "prob": "0.5"
          }
        }
      ]
    },
    {
      "unary_neg": {
        "binary_and": [
          {
            "atomic_tuple_event": {
              "id": "3",
              "prob": "0.1"
            }
          },
          {
            "atomic_tuple_event": {
              "id": "4",
              "prob": "0.8"
            }
          }
        ]
      }
    }
  ]
}

```

Figure 15.7: Lineage formula $(e_1 \odot e_2) \vee \neg(e_3 \wedge e_4)$ encoded as a JSON-object

Lineage construction: nested lineage formulas

The next tested method transformed a set of JSON-encoded lineage formulas produced by the relational database layer into a set of formula trees consisting of atomic tuple events and logical operators. Thereby, the JSON-encoded lineage formulas already embodied the nested structures of the final lineage formulas. Accordingly, we could directly transform them into a set of formula trees very efficiently by highly optimized JSON standard libraries.

Example 15.6 (JSON-encoding of nested lineage formulas). *Figure (15.7) on Page (157) depicts the nested lineage formula*

$$(e_1 \odot e_2) \vee \neg(e_3 \wedge e_4)$$

with

$$\mathbf{P}(e_1) = 0.3, \quad \mathbf{P}(e_2) = 0.5, \quad \mathbf{P}(e_3) = 0.1 \quad \text{and} \quad \mathbf{P}(e_4) = 0.8$$

represented as JSON-object.

Lineage construction: factor networks

In Section (8.4), we presented in Figure (8.5) and (8.6) on Page (58) and (60) two sets of tables storing two networks consisting of factors and Boolean functions. Both formalisms are able to represent lineage formulas. Such sort of tables were already created within the relational database layer, see Section (13.5).

In the subsequent lineage construction phase, we had to set up the intended networks within

our probabilistic query engine. More concretely, we directly transferred the generated tables into a set of so-called *factor catalogs*¹ containing lists of factors with pointers to their respective operands.

Lineage construction: vertical lineage construction

The last technique under investigation was our vertical construction algorithm. In the previous sections of this chapter, we presented the basic form of our algorithm. In addition, we also describe in Section (16.2) an adjusted variant processing decomposed event relations as input. Both variants were exploited in our experiments, see classification of all test queries in Section (13.5).

Measured data

In our second series of experiments, we were particularly interested in the following aspects:

- the accumulated number of all atomic tuple events used to build our lineage formulas per approach, see rows labeled with *lineage nodes* in Figure (15.8) and (15.9), and
- the mean of the processing times (wall-clock) of 100 runs for constructing all lineage formulas within our probabilistic query engine per approach, see the rows labeled with *lineage* in Figure (15.8) and (15.9).

To visualize the interrelations between the processing parts performed within relational database layer and probabilistic query engine, we recap in Figure (15.8) and (15.9) the measured data already known from Section (13.5).

Experimental observations

In the following, we discuss the properties and computation times we measured in our second series of experiments.

Measured properties: lineage nodes in the probabilistic query engine

The measured number of lineage nodes directly corresponds to the cumulative lengths of all constructed lineage formulas.

- **safe plans:** Safe plans, as used in MystiQ, did not construct any lineage nodes, since all answer probabilities were calculated in the relational database layer.
- **DNF lineage:** The unfolded lineage formulas in DNF were clearly longer than their nested counterparts.
- **nested lineage:** The number of nodes for nested lineage formulas directly gave the lengths of lineage formulas constructed by Lemma (6.1) on Page (35).
- **networks:** The sizes of factor networks were smaller than nested lineage formulas, because network factors could be already shared between different lineage formulas.
- **vertical:** Since we used the same query plans as for nested lineage, we also achieved identical lineage node counts.

Measured computation times: lineage construction within the probabilistic query engine

- **safe plans:** No lineage construction time was measured for safe plans.
- **DNF lineage:** If the number of input tuples was relatively low (TPCH-Q1 and TPCH-Q2), the pure lineage construction times were the fastest of all approaches. But in the majority of queries, the large number of input tuples influenced the measured construction times negatively, e.g., IMDB-Q3, TPCH-Q3, and TPCH-Q4.

¹We use this term in accordance with our own terminology of Chapter (19).

Measured properties: lineage construction						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	RDBMS relation(s)	1	1	1	13	7
	RDBMS tuples	5,473	91,406	5,473	369,212	5,480
	lineage nodes	0	548,436	372,678	104,212	372,678
IMDB-Q2	RDBMS relation(s)	1	1	1	7	5
	RDBMS tuples	30	106,033	30	37,298	3,400
	lineage nodes	0	424,132	408,280	35,298	408,280
IMDB-Q3	RDBMS relation(s)	1	1	1	8	5
	RDBMS tuples	1,516	940,008	1,516	215,636	1,520
	lineage nodes	0	3,760,032	296,748	80,636	296,748
IMDB-Q4	RDBMS relation(s)	1	-	1	10	1
	RDBMS tuples	2,216	-	2,216	926,081	2,995
	lineage nodes	0	-	38,572	26,382	38,572
IMDB-Q5	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	5,535	-	5,535	1,744,059	6,257
	lineage nodes	0	-	139,188	98,151	139,188
IMDB-Q6	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	141	-	141	655,243	1,480
	lineage nodes	0	-	5,518	4,356	5,518

Measured computation times: lineage construction						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	relational	10.36	1.612	16.806	3.301	1.385
	lineage	0.0	1.659	0.707	2.337	1.56
	total	10.36	3.271	17.513	5.638	2.945
IMDB-Q2	relational	1.109	0.428	17.971	0.368	0.267
	lineage	0.0	1.504	0.532	0.175	0.2
	total	1.109	1.933	18.503	0.543	0.467
IMDB-Q3	relational	33.898	3.958	190.858	7.075	2.839
	lineage	0.0	8.249	0.359	9.289	0.297
	total	33.898	12.207	191.218	16.364	3.136
IMDB-Q4	relational	13.31	-	14.77	3.699	0.097
	lineage	0.0	-	0.048	4.834	0.336
	total	13.31	-	14.817	8.532	0.433
IMDB-Q5	relational	32.182	-	52.796	15.478	0.593
	lineage	0.0	-	0.201	64.724	1.578
	total	32.182	-	52.998	80.202	2.171
IMDB-Q6	relational	14.48	-	19.746	5.375	0.455
	lineage	0.0	-	0.008	4.227	0.037
	total	14.48	-	19.754	9.602	0.491

Figure 15.8: IMDB queries: measured properties and computation times for lineage construction within a probabilistic query engine

- **nested lineage:** The construction times of nested lineage formulas were fast, because we could directly exploit very efficient standard libraries for decoding the delivered JSON structures.
- **networks:** The times measured for building factor networks suffered from their large input sizes and showed the minimization effort for identifying all necessary lineage subformulas.
- **vertical:** Our **vlc**-algorithm could mainly benefit from its flexibility of processing different event relation forms with moderate input sizes. If we consider the pure construction time for a single input tuple from \mathbf{E}_Q , we observed a relatively high value.

Measured properties: lineage construction						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	RDBMS relation(s)	1	1	1	6	1
	RDBMS tuples	125	1,725	125	154,047	1,725
	lineage nodes	0	5,175	4,718	2,014	4,718
TPCH-Q2	RDBMS relation(s)	1	1	1	10	1
	RDBMS tuples	961	14,752	961	176,761	14,752
	lineage nodes	0	59,008	43,054	31,234	43,054
TPCH-Q3	RDBMS relation(s)	1	1	1	12	6
	RDBMS tuples	38	890,072	38	239,647	11,453
	lineage nodes	0	4,450,360	72,488	60,904	72,488
TPCH-Q4	RDBMS relation(s)	1	1	1	11	6
	RDBMS tuples	14	1,277,028	14	361,648	1,125
	lineage nodes	0	6,385,140	159,204	99,238	159,204
TPCH-Q5	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	99	-	99	151,520	1,520
	lineage nodes	0	-	271,788	51,788	271,788
TPCH-Q6	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	63	-	63	348,128	5,045
	lineage nodes	0	-	233,362	128,974	233,362
TPCH-Q7	RDBMS relation(s)	1	-	1	14	7
	RDBMS tuples	27	-	27	493,531	2,166
	lineage nodes	0	-	100,180	66,233	100,180

Measured computation times: lineage construction						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	relational	12.332	0.112	15.082	2.173	0.113
	lineage	0.0	0.015	0.036	3.119	0.144
	total	12.332	0.127	15.118	5.292	0.257
TPCH-Q2	relational	8.296	0.489	10.96	2.353	0.488
	lineage	0.0	0.026	0.126	1.703	0.946
	total	8.296	0.514	11.086	4.055	1.434
TPCH-Q3	relational	7.363	2.411	12.847	3.721	1.683
	lineage	0.0	8.407	0.109	1.891	0.542
	total	7.363	10.818	12.956	5.613	2.225
TPCH-Q4	relational	14.536	2.649	55.117	5.88	1.207
	lineage	0.0	14.652	0.324	7.535	0.186
	total	14.536	17.301	55.442	13.415	1.393
TPCH-Q5	relational	1.877	-	4.674	0.81	0.37
	lineage	0.0	-	0.438	1.009	0.113
	total	1.877	-	5.111	1.819	0.483
TPCH-Q6	relational	5.675	-	12.307	3.65	0.898
	lineage	0.0	-	0.278	1.387	0.498
	total	5.675	-	12.586	5.037	1.396
TPCH-Q7	relational	9.534	-	15.386	3.862	1.381
	lineage	0.0	-	0.122	1.979	0.347
	total	9.534	-	15.508	5.841	1.727

Figure 15.9: TPC-H queries: measured properties and computation times for lineage construction within a probabilistic query engine

15.3 Summary

In this chapter, we described the vertical construction principle of our central algorithm. It performs complete path insertions instead of a stepwise connection of formula subtrees. We also showed its pragmatic benefits experimentally.

In the motivation presented in Part (III), we stated four design goals for lineage construction,

which *all* can be satisfied by our approach:

- **Design goal (1): full relational algebra support:** neither the generation rules for event relations nor the **vlc**-algorithm exclude any relational operators from our query language,
- **Design goal (2): unlimited lineage formula lengths:** our lineage construction is not limited by the maximal sizes of specific data types,
- **Design goal (3): native relational operators and data types:** our rules for generating event relations are built by relational standard operators, and
- **Design goal (4): relational result set sizes as deterministic case:** our event relations just grow polynomially in database size.

Chapter 16

Advanced aspects of vertical lineage construction

In this chapter, we present the following advanced topics:

- Section (16.1): a series of lemmas proving our vertical lineage construction algorithm and
- Section (16.2): a further variant of the vertical lineage construction algorithm processing decomposed event relations as an input.

16.1 Correctness of vertical lineage construction

In this section, we verify our vertical construction algorithm. Therefore, we connect the constructed formula tree of $\mathbf{vlc}(t, \mathbf{E}_Q, Q)$ to our theoretical model that has been already proved in Chapter (10). Under the hood, our algorithm executes the three main transformation steps of Figure (10.1) on Page (80):

- processing an equivalent domain calculus query Q^c instead of an algebra query Q ,
- transforming the first-order condition Φ^t of Q^c into an equivalent propositional relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$, and
- mapping a relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ to an alternative lineage formula φ^t .

Example 16.1 (Main transformation chain). *On the basis of our example query*

$$Q_{\bowtie} = \pi_A((R_1 \bowtie R_2) \bowtie (R_1 \bowtie R_3)) \equiv \{t \mid \Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^t \rangle}^t\} = Q_{\bowtie}^c,$$

we exemplify our basic transformation chain by constructing the lineage formula $\varphi_{\bowtie}^{(1)}$ with its relevant domain $\mathbf{D}^{(1)} = \{(3)_{x_B}, (4)\}$:

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x} | \mathbf{D}^{(1)} \rangle}^{(1)} &= \bigvee_{d_{x_B} \in \mathbf{D}^{(1)}} \phi_{\bowtie, \langle x_B | d \rangle}^{(1)} \\ &\equiv \bigvee_{d_{x_B} \in \mathbf{D}^{(1)}} ((R_1(1) \wedge R_2(1, x_B)) \wedge (R_1(1) \wedge R_3(1, x_B)))_{\langle x_B | d_{x_B} \rangle} \\ &\equiv ((R_1(1) \wedge R_2(1, 3)) \wedge R_3(1, 3)) \vee ((R_1(1) \wedge R_2(1, 4)) \wedge R_3(1, 4)) \\ &\mapsto ((X_{R_1(1)}(W) \wedge X_{R_2(1,3)}(W)) \wedge X_{R_3(1,3)}(W)) \vee \\ &\quad ((X_{R_1(1)}(W) \wedge X_{R_2(1,4)}(W)) \wedge X_{R_3(1,4)}(W)) \\ &\equiv ((e_1 \wedge e_3) \wedge e_5) \vee ((e_1 \wedge e_4) \wedge e_6) \\ &= \varphi_{\bowtie}^{(1)}. \end{aligned}$$

The proposed transformation chain is thereby supported by the following construction and simplification mechanisms:

- atomic-tuple-event-wise processing of a relevant domain value d ,
- simplifying lineage subformulas that are substituted by domain values ranging over an entire domain,
- simplifying lineage subformulas that only consist of the impossible tuple events F , and
- simplifying lineage subformulas that are derived from unsatisfied selection operations.

Atomic-tuple-event-wise processing of a relevant domain value d

In Section (10), we described how we set up a relevant condition of the form $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ by connecting all substituted condition structures $\phi_{\langle \bar{x} | d \rangle}^t$ domain-value-by-domain-value:

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \overline{\bigotimes_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x} | d \rangle}^t.$$

For each substituted condition structure $\phi_{\langle \bar{x} | d \rangle}^t$, we replaced all variables \bar{x} in ϕ^t by their respective values of d .

Our algorithm works differently. Instead of inserting values for variables, it adds atomic tuple events $d.e$ into $\phi_{\langle \bar{x} | d \rangle}^t$. Its main loop builds $\phi_{\langle \bar{x} | d \rangle}^t$ atomic-tuple-event-wise, i.e., $d.e$ -by- $d.e$, see Line (5) in Algorithm (3) on Page (149).

Obviously, we can significantly minimize substitution operations, when we map $R(\bar{x}, \bar{c})$ to e_t in one step instead of substituting a set of variables \bar{x} in $R(\bar{x}, \bar{c})$ value-by-value.

Lemma 16.1 (Processing of a relevant domain value d). *Let Q be an algebra query with its equivalent domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \overline{\bigotimes_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x} | d \rangle}^t\}$$

and let \mathbf{E}_Q be its event relation. If $R(\bar{x}, \bar{c})$ is a relation predicate of ϕ^t and φ_Q^t is the lineage formula mapped from $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$, then

$$(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \overline{\bigotimes_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x} | d \rangle}^t) \Rightarrow (\varphi_Q^t \equiv \overline{\bigotimes_{d \in \mathbf{D}^t}} \phi_{\langle R(\bar{x}, \bar{c}) | d.e \rangle}^t)$$

holds, given that the atomic tuple event $d.e$ comes from the column of \mathbf{E}_Q with label $R(\bar{x}, \bar{c})$.

Proof. Thanks to Definition (13.1) on Page (124), Lemma (13.3) on Page (125), and Lemma (14.3) on Page (138), we can be sure that a row of \mathbf{E}_Q contains *all* atomic tuple events that are required for creating the lineage subformula based on the specific substituted condition structure $\phi_{\langle \bar{x} | d \rangle}^t$. Thereby, each atomic tuple event $d.e$ has originated in a unique way from a relation predicate $R(\bar{x}, \bar{c})$. Our algorithm exploits this direct connection between an atomic tuple event $d.e$ and its relation predicate $R(\bar{x}, \bar{c})$ by putting $d.e$ directly in its positions in $\phi_{\langle \bar{x} | d \rangle}^t$. Those positions are indicated through the corresponding relation predicate within the underlying condition structure.

This is possible *without* considering any variables, since we always navigate within only *one* specific condition structure $\phi_{\langle \bar{x} | d \rangle}^t$ determined by the current domain value d . In other words, we simply lead all atomic tuple events to their positions marked by their relation predicates after

choosing the correct substituted condition structure $\phi_{\langle \bar{x}|d \rangle}^t$ from the leading sequence of n-ary operations.

This mechanism is equivalent to a replacement of each variable within a relation predicate $R(\bar{x}, \bar{c})$ by the domain values of d and a subsequent mapping from substituted relation predicates $R(\bar{c}, _)$ to atomic tuple events. \square

Example 16.2 (Processing of a relevant domain value d). *Let us consider our example query*

$$Q_{\bowtie} = \pi_A((R_1 \bowtie R_2) \bowtie (R_1 \bowtie R_3)) \equiv \{t \mid \Phi_{\bowtie, \langle \bar{x}|\mathbf{D}^t \rangle}^t\} = Q_{\bowtie}^c$$

in conjunction with its event relation

$$\mathbf{E}_{\bowtie} = \left\{ \underbrace{(1, 3, e_1, e_3, e_5)_{(t_{x_A}, x_B, R_1(t_{x_A}), R_2(t_{x_A}, x_B), R_3(t_{x_A}, x_B))}}_{d_1}, \underbrace{(1, 4, e_1, e_4, e_6)}_{d_2} \right\}$$

generated in Example (13.5) on Page (129). When we directly replace relation predicates by atomic tuple events within one specific condition structure, we do not need to care for variables:

$$\begin{aligned} \Phi_{\bowtie, \langle \bar{x}|\mathbf{D}^{(1)} \rangle}^{(1)} &= \bigvee_{d \in \mathbf{D}^{(1)}} \phi_{\bowtie, \langle x_B|d_{x_B} \rangle}^{(1)} \\ &\equiv \bigvee_{d \in \mathbf{E}_{\bowtie}} \underbrace{\left((R_1(t_{x_A}) \wedge R_2(t_{x_A}, x_B)) \wedge R_3(t_{x_A}, x_B) \right)}_{\substack{\text{positions of } R_1(t_{x_A}), R_2(t_{x_A}, x_B) \text{ and } R_3(t_{x_A}, x_B) \\ \text{are unique for a specific } d}} \langle x_B|d_{x_B} \rangle \\ &\mapsto \underbrace{\left((R_1(t_{x_A}) \wedge R_2(t_{x_A}, x_B)) \wedge R_3(t_{x_A}, x_B) \right)}_{\substack{\text{initiated by atomic tuple events of } \mathbf{atomicTupleEvents}(d_1)}} \langle R_1(t_{x_A}), R_2(t_{x_A}, x_B), R_3(t_{x_A}, x_B) | d_1.e_1, d_1.e_3, d_1.e_5 \rangle \bigvee \\ &\quad \underbrace{\left((R_1(t_{x_A}) \wedge R_2(t_{x_A}, x_B)) \wedge R_3(t_{x_A}, x_B) \right)}_{\substack{\text{initiated by atomic tuple events of } \mathbf{atomicTupleEvents}(d_2)}} \langle R_1(t_{x_A}), R_2(t_{x_A}, x_B), R_3(t_{x_A}, x_B) | d_2.e_1, d_2.e_4, d_2.e_6 \rangle \\ &\equiv ((e_1 \wedge e_3) \wedge e_5) \bigvee ((e_1 \wedge e_4) \wedge e_6) \\ &= \varphi_{\bowtie}^{(1)}. \end{aligned}$$

For optimization reasons, we exclude the impossible tuple event F from the main loop, see Line (4) in Algorithm (3) on Page (149). Instead, we treat it by the mechanisms explained in the following subsections.

Simplifying lineage subformulas that are substituted by domain values ranging over an entire domain

In Definition (11.4) on Page (104), we specified that $d_{\bar{y}} = (_)$ stands for the relevant domain $\mathbf{D}_{\bar{y}}^t = \mathbf{D}_{\bar{y}}^t$. In this case, a relevant condition $\Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t$ consists of an (infinite) list of connected substituted condition structures, as already shown in Example (13.3) on Page (127).

Although $d = (_)_{\bar{y}}$ represents a set of substituted condition structures, the main loop of our algorithm processes a domain value $d = (_)_{\bar{y}}$ only once.

Lemma 16.2 (Simplifying lineage subformulas that are substituted by domain values ranging over an entire domain). *Let Q be an algebra query with its equivalent domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}^t} \phi_{\langle \bar{x}|d \rangle}^t\}$$

and let $R(\bar{c}, (_)_{\bar{y}})$ be a relation predicate used within $\phi_{\langle \bar{x}|d \rangle}^t$.

If we split the overall variable set \bar{x} into $\bar{x} = (\bar{y} \dot{\cup} \bar{z})$, we only have to consider $\mathbf{D}_{\bar{z}}^t$ instead of \mathbf{D}^t :

$$\Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t \equiv \overline{\bigvee_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x}|d \rangle}^t \equiv \overline{\bigvee_{d_{\bar{z}} \in \mathbf{D}_{\bar{z}}^t}} (\phi_{\langle R(\bar{c}, (_)_{\bar{y}})|F \rangle}^t)_{\langle \bar{z}|d_{\bar{z}} \rangle}.$$

Proof. In Lemma (14.3) on Page (138), we already proved that the relation predicate $R(\bar{c}, (_)_{\bar{y}})$ within a condition structure $\phi_{\langle \bar{x}|d \rangle}^t$ can be simplified with F . Then, all condition structures substituted by domain values of $d = (\bar{c}, (_)_{\bar{y}})$ have identical forms. In fact, we eliminate the parts, where the substituted condition structures differ from each other.

As a result, we achieve a set of identically substituted condition structures connected by n-ary disjunction/conjunction operations. Then, we can apply the logical law of idempotence and reduce all identical condition structures to a single one, which is eventually processed by our algorithm. \square

Example 16.3 (Simplification of lineage subformulas substituted by domain values ranging over an entire domain). We continue here Example (13.4) on Page (128). There, we simplified the relation predicate $R_3(2, _)$ within

$$\begin{aligned} \Phi_{\langle \bar{x}|\mathbf{D}^{(2)} \rangle}^{(2)} &\equiv \dots \otimes \underbrace{(R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, _) \wedge (2 = 1))}_{\phi_{\langle \bar{x}_B, x'_B | d_{10} \rangle}^{(2)}} \otimes \dots \\ &\equiv \dots \otimes \underbrace{(\phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} \otimes \dots \otimes \phi_{\langle \bar{x}_B, x'_B | d''_{10} \rangle}^{(2)})}_{\phi_{\langle \bar{x}_B, x'_B | d_{10} \rangle}^{(2)}} \otimes \dots \end{aligned}$$

to

$$\begin{aligned} \phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, 4) \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)) \\ \phi_{\langle \bar{x}_B, x'_B | d''_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, 5) \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)) \\ \phi_{\langle \bar{x}_B, x'_B | d'''_{10} \rangle}^{(2)} &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, 6) \wedge (2 = 1)) \\ &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)) \\ \phi_{\langle \bar{x}_B, x'_B | d''''_{10} \rangle}^{(2)} &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, 7) \wedge (2 = 1)) \\ &\equiv (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)). \end{aligned}$$

Consequently, we achieve identically substituted condition structures

$$\phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} = \dots = \phi_{\langle \bar{x}_B, x'_B | d''''_{10} \rangle}^{(2)} = (R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1)),$$

which can be expressed as

$$\begin{aligned} \Phi_{\langle \bar{x}|\mathbf{D}^{(2)} \rangle}^{(2)} &\equiv \dots \otimes (\phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} \otimes \dots \otimes \phi_{\langle \bar{x}_B, x'_B | d''''_{10} \rangle}^{(2)}) \otimes \dots \\ &\equiv \dots \otimes (\phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} \otimes \dots \otimes \phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)}) \otimes \dots \\ &\equiv \dots \otimes \phi_{\langle \bar{x}_B, x'_B | d'_{10} \rangle}^{(2)} \otimes \dots \end{aligned}$$

$$\begin{aligned} &\equiv \dots \otimes ((R_1(2) \wedge R_3(2, 5)) \wedge \neg(F \wedge (2 = 1))) \otimes \dots \\ &\equiv \dots \otimes ((e_2 \wedge e_7) \wedge \neg(F \wedge F)) \otimes \dots \end{aligned}$$

without involving $R_3(2, _)$ any more.

Simplifying lineage subformulas that only consist of impossible tuple events

As mentioned earlier, our construction algorithm only adds paths needed to bring a certain atomic tuple event $d.e$ to its correct positions within the evolving formula tree. Since the impossible tuple event F is excluded from all paths insertions (Line (4) of Algorithm (3) on Page (149)), there are leaves and paths which are purposely left out from creating. Subformulas only consisting of paths with \boxed{F} -leaves are simplified to a single node with type \boxed{F} .

Please be aware that such a simplification is not valid in general. The subcondition $F \vee \neg(F)$, for instance, is not equivalent to F . Therefore, we prove our applied simplification.

Lemma 16.3 (Simplification of lineage subformulas only consisting of impossible tuple events).
Let Q be an algebra query with its equivalent domain calculus query:

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}^t} \phi_{\langle \bar{x} \mid d \rangle}^t\}.$$

If ϕ^F denotes a propositional formula where all involved atomic tuple events are given by F , then

$$(\phi^F \text{ is subformula of } \phi^t) \Rightarrow (\phi^t \equiv \phi_{\langle \phi^F \mid F \rangle}^t)$$

holds.

Proof. We prove our proposition by an induction proof over the number of operations n in Q and the following logical equivalences inferred from the MAC rules of Definition (10.1) on Page (84):

$$\begin{aligned} Q = R &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (R(\bar{c}, _))_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F \equiv F \\ Q = \sigma_F(Q_1) &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\phi_{Q_1}^t \wedge F)_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F \wedge F \equiv F \\ Q = \pi_{\mathcal{A}}(Q_1) &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\bigvee(\phi_{Q_1}^t))_{\langle R(\bar{c}, _) \mid F \rangle} \equiv \bigvee(F) \equiv F \\ Q = Q_1 \bowtie Q_2 &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\phi_{Q_1}^t \wedge \phi_{Q_2}^t)_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F \wedge F \equiv F \\ Q = Q_1 \cup Q_2 &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\phi_{Q_1}^t \vee \phi_{Q_2}^t)_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F \vee F \equiv F \\ Q = Q_1 \setminus Q_2 &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\phi_{Q_1}^t \wedge \neg(\phi_{Q_2}^t))_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F \wedge \neg(F) \equiv F \\ Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1) &: \phi_{Q, \langle R(\bar{c}, _) \mid F \rangle}^t \equiv (\phi_{Q_1, \langle \mathcal{A} \mid \mathcal{B} \rangle}^t)_{\langle R(\bar{c}, _) \mid F \rangle} \equiv F_{\langle \mathcal{A} \mid \mathcal{B} \rangle} \equiv F. \end{aligned}$$

Thereby, the relation rule $Q = R$ forms the induction basis ($n = 1$) and the remaining rules take effect for queries with $(n + 1)$ operators assuming that our proposition has already been proven for the subqueries Q_1 and Q_2 . \square

Example 16.4 (Simplification of lineage subformulas only consisting of impossible tuple events).
We continue to explore our substituted condition structure $\phi_{\setminus \langle \bar{x} \mid d_{10} \rangle}^{(2)}$, which is part of the relevant

condition $\Phi_{\setminus, \langle \bar{x} | \mathbf{D}^{(2)} \rangle}^{(2)}$. We already know from Example (13.4) on Page (128) that

$$\begin{aligned}\phi_{\setminus, \langle \bar{x} | d_{10} \rangle}^{(2)} &= (R_1(2) \wedge R_3(2, 5)) \wedge \neg(R_3(2, _) \wedge (2 = 1)) \\ &\equiv (e_2 \wedge e_7) \wedge \neg(F \wedge F).\end{aligned}$$

If we compare the last form of $\phi_{\setminus, \langle \bar{x} | d_{10} \rangle}^{(2)}$ with the constructed formula tree of Figure (15.6) on Page (156), we recognize that the subformula $F \wedge F$ is not constructed, since our algorithm has not inserted any path with a leaf $d_{10}.F$. Thus, the subtree representing

$$R_3(2, _) \wedge (2 = 1) \quad \text{and} \quad R_3(2, _) \wedge (2 = 1) \equiv F \wedge (2 = 1) \equiv F$$

was not generated. Instead, it was implicitly simplified to a single node with type \boxed{F} . This node was created by its parent node with type $\boxed{\wedge \neg}$. It kept its initial node type \boxed{F} until the end of the overall construction.

Simplification of lineage subformulas derived from unsatisfied selection operations

In Line (21) of Algorithm (4) on Page (150), our algorithm evaluates a selection condition F based on the current processed domain value d . If d is not able to fulfill the selection condition, there is obviously no need to construct the *conjunctively* connected subformula any further.

Lemma 16.4 (Simplification of lineage subformulas derived from unsatisfied selection operations). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \overline{\bigvee \bigotimes} \phi_{\langle \bar{x} | d \rangle}^t\}.$$

Since the MAC rule for the selection operator of Definition (10.1) on Page (84) implies the condition structure

$$Q = \sigma_F(Q_1) \quad : \quad \phi_Q^t \equiv \phi_{Q_1}^t \wedge F,$$

then

- *Property (A):*

$$(F(d) \equiv F) \Rightarrow (\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv F) \quad \text{and}$$

- *Property (B):*

$$(F(d) \equiv T) \Rightarrow (\phi_{Q, \langle \bar{x} | d \rangle}^t \equiv \phi_{Q_1}^t)$$

hold.

Proof. The selection condition F is *conjunctively* combined with the subcondition structure $\phi_{Q_1, \langle \bar{x} | d \rangle}^t$.

- *Property (A):* Thus, we can obviously neglect the construction of $\phi_{Q_1, \langle \bar{x} | d \rangle}^t$, if $F(d)$ is not fulfilled:

$$\phi_{Q, \langle \bar{x} | d \rangle}^t = (\phi_{Q_1}^t \wedge F)_{\langle \bar{x} | d \rangle} \equiv \phi_{Q_1, \langle \bar{x} | d \rangle}^t \wedge F(d) \equiv \phi_{Q_1, \langle \bar{x} | d \rangle}^t \wedge F \equiv F.$$

If the selection condition $F(d)$ is not satisfied in Line (21) of Algorithm (4) on Page (150), the type of the current node is set to \boxed{F} .

- Property (B): When $F(d)$ is fulfilled, we can simplify $\phi_{Q, \langle \bar{x}|d \rangle}^t$ to

$$\phi_{Q, \langle \bar{x}|d \rangle}^t = (\phi_{Q_1}^t \wedge F)_{\langle \bar{x}|d \rangle} \equiv \phi_{Q_1, \langle \bar{x}|d \rangle}^t \wedge F(d) \equiv \phi_{Q_1, \langle \bar{x}|d \rangle}^t \wedge \top \equiv \phi_{Q_1, \langle \bar{x}|d \rangle}^t.$$

In that case, our algorithm calls **insertPath** with the same node and the next algebra operator Q_1 , see Line (22) of Algorithm (4) on Page (150). There is no specific node type materialized for the selection operation $\sigma_F(Q_1)$.

□

Example 16.5 (Simplification of lineage subformulas derived from unsatisfied selection operations). In Figure (15.6) and (17.6) on Page (156) and Page (181), we see two examples of the simplification of subformulas derived from a selection operation.

Correctness and complexity proof for vertical lineage construction

Next, we give the final correctness and complexity proofs for our proposed vertical lineage construction algorithm. They summarize the lemmas already presented in the last sections.

Theorem 16.1 (Correctness of vertical lineage construction algorithm). *Let Q be an algebra query with its equivalent domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x}|\mathbf{D} \rangle}^t\} \equiv \{t \mid \overline{\bigotimes_{d \in \mathbf{D}}} \phi_{\langle \bar{x}|d \rangle}^t\}$$

and let \mathbf{E}_Q be its event relation. When $\mathbf{vlc}(t, \mathbf{E}_Q, Q)$ constructs a lineage formula for a given tuple t , i.e., $\varphi^t := \mathbf{vlc}(t, \mathbf{E}_Q, Q)$, we can determine the answer probability of t as

$$\mathbf{P}(t \in Q) = \mathbf{P}(\varphi^t).$$

Proof. We already proved in (11.2) that

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x}|\mathbf{D}^* \rangle}^t).$$

Then, the **vlc**-algorithm works correctly, if it returns a lineage formula

$$\varphi^t := \mathbf{vlc}(t, \mathbf{E}_Q, Q) \quad \text{with} \quad \mathbf{P}(\varphi^t) = \mathbf{P}(\Phi_{\langle \bar{x}|\mathbf{D}^* \rangle}^t).$$

As seen in Algorithm (3) and (4) on Page (149) and (150), we build a resulting lineage formula φ^t in the form of a formula tree. The order and types of all path nodes are determined by the implicitly given transformation mapping between algebra operators, relevant conditions, and lineage formulas, see Lemma (10.1) and (10.2) on Page (84) and (87), and Theorem (10.1) on Page (94). It is directly implemented by Lemma (16.1) on Page (164).

In addition to this main transformation, the underlying relevant condition is simplified by techniques proved in Lemma (16.2), (16.3), and (16.4) on Page (165), (167), and (168). □

Lemma 16.5 (Complexity of vertical lineage construction algorithms). *Let Q be an algebra query with its domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x}|\mathbf{D} \rangle}^t\} \equiv \{t \mid \overline{\bigotimes_{d \in \mathbf{D}}} \phi_{\langle \bar{x}|d \rangle}^t\}.$$

It is applied on a probabilistic database **pdb** consisting of all possible tuple combinations of $R_1(W^{max}), \dots, R_m(W^{max})$. Then, the number of steps needed for determining the query result $Q(\mathbf{pdb})$ is bounded by

$$|Q|^2 * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|},$$

if we use our vertical lineage construction algorithm laid out in Algorithm (3) and (4) on Page (149) and (150).

Proof. From relational algebra, we know that the size of our event relation \mathbf{E}_Q of Definition (13.1) on Page (124) is limited by $\max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$. Moreover, it is clear that the number of relation predicates involved in an underlying condition structure ϕ^t is at most as large as the query length of Q , since they are mapped from the relations of Q . Considering that each relation predicate forms a leaf, we have no more than $|Q|$ paths to create per one row of \mathbf{E}_Q .

Since the length of a single path cannot exceed the size of an underlying condition structure ϕ^t , we achieve a maximum of $|Q|^2$ construction steps per domain value. In total, that leads to a boundary of

$$|Q|^2 * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|}$$

steps for processing all domain values of \mathbf{E}_Q . □

16.2 Vertical lineage construction algorithm for decomposed event relations

In Algorithm (5) and (6) on Page (170) and (171), we provide an adjusted version of our vertical construction algorithm. Instead of composed event relations, as introduced in Chapter (13), this version works on decomposed event relations described in Section (14.3). Please recall that our idea of processing decomposed event relations consists of:

- a query in the *normal form* Q^d , where each relation operator is wrapped in a subquery of the form $\pi_A(\sigma_F(R))$, see Example (14.3) on Page (145),
- a set of *event source relations* $\mathbf{S}_{R(\bar{x}, t_{\bar{y}})}$ that contains all required atomic tuple events, see Definition (14.2) and Example (14.5) on Page (146) and (146), and
- a single *join relation* \mathbf{J}_Q , which only comprises domain values that are necessary to compose all event source relations, see Definition (14.1) and Example (14.4) on Page (145) and (145).

Algorithm 5: $\text{vlc}(t, \mathbf{J}_Q, Q)$

```

1 rootNode :=  $\boxed{\mathbf{F}}$ ;
2 foreach  $d \in \mathbf{J}_Q$  do
3   if  $d_{(\text{vars}(\text{head}(Q)))} = t$  then
4     foreach  $R(\bar{x}, t_{\bar{y}}) \in \text{relPredicates}(\phi_Q^t)$  do
5       insertPaths( $d, R(\bar{x}, t_{\bar{y}}), Q, \text{rootNode}$ );
6     end
7   end
8 end
9 return rootNode;

```

Algorithm 6: $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q, \text{node})$

```

1  switch  $Q$  do
2    case  $Q = \pi_{\mathcal{A}}(\sigma_F(R))$ 
3      |  $\text{node} := \boxed{\bigvee}$ ;
4      | foreach  $(\hat{d}.e \in \mathbf{S}_{R(\bar{x}, t_{\bar{y}})})$  do
5        |   if  $(\hat{d}_{\bar{y}} = d_{\bar{y}}) \wedge (F(\hat{d}) \equiv T)$  then
6          |      $\text{node.children}[\hat{d}] := \boxed{e}$ ;
7          |   end
8      | end
9      | return;
10   case  $Q = \pi_{\mathcal{A}}(Q_1)$ 
11     |  $\text{node} := \boxed{\bigvee}$ ;
12     |  $\text{key} := d_{\text{vars}(\text{head}(Q_1) \setminus \mathcal{A})}$ ;
13     |  $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q_1, \text{createIfAbsent}(\text{node.children}[\text{key}]))$ ;
14   case  $Q = Q_1 \Theta Q_2$  with  $\Theta \in \{\bowtie, \cup, \setminus\}$ 
15     |  $\text{node} := \begin{cases} \boxed{\wedge} & \text{if } \Theta = \bowtie \\ \boxed{\vee} & \text{if } \Theta = \cup \\ \boxed{\wedge \neg} & \text{if } \Theta = \setminus \end{cases}$ 
16     | if  $((\text{relPredicatesMappedFrom}(R(\bar{x}, t_{\bar{y}})) \cap \text{relPredicates}(\phi_{Q_1}^t)) \neq \emptyset)$  then
17       |    $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q_1, \text{createIfAbsent}(\text{node.leftChild}))$ ;
18       |   if  $(\text{absent}(\text{node.rightChild}))$  then
19         |      $\text{node.rightChild} := \boxed{F}$ ;
20       |   end
21     | end
22     | if  $((\text{relPredicatesMappedFrom}(R(\bar{x}, t_{\bar{y}})) \cap \text{relPredicates}(\phi_{Q_2}^t)) \neq \emptyset)$  then
23       |    $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q_2, \text{createIfAbsent}(\text{node.rightChild}))$ ;
24       |   if  $(\text{absent}(\text{node.leftChild}))$  then
25         |      $\text{node.leftChild} := \boxed{F}$ ;
26       |   end
27     | end
28   case  $Q = \sigma_F(Q_1)$ 
29     | if  $(F(d) \equiv T)$  then
30       |    $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q_1, \text{node})$ ;
31     | else
32       |    $\text{node} := \boxed{F}$ ;
33     | end
34   case  $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$ 
35     |  $\text{insertPaths}(d, R(\bar{x}, t_{\bar{y}}), Q_1, \text{node})$ ;
36   endsw
37 endsw

```

Chapter 17

Lineage optimization

In this chapter, we address optimization strategies for probabilistic database systems. We particularly develop a novel approach, which is capable of an orthogonal combination of optimizations performed within the relational database layer and probabilistic query engine. Following topics are discussed in detail:

- Section (17.1): motivation and overview of existing state-of-the-art optimization techniques,
- Section (17.2): our key idea of a *decoupled lineage optimization* integrated within our vertical lineage construction framework.

17.1 Motivation and existing lineage optimization approaches

In principle, there are two different classes of approaches optimizing the query evaluation process for probabilistic databases:

- Query plan optimization: First, we will shortly describe a method that rewrites and optimizes the given input query into a relational *safe plan*. By means of a safe plan, we can either push the entire probability computation into the relational database layer or create tractable lineage formulas that are easy to evaluate within a probabilistic query engine.
- Lineage formula optimization: Secondly, we will sketch an optimization approach working directly on already generated lineage formulas.

In order to exemplify the core ideas of both basic techniques we revisit our IMDB example already known from Example (7.1) and (9.1) on Page (51) and (64).

Example 17.1 (IMDB example (revisit)). *In Example (7.1) on Page (51), we defined a small TID database, which embodies a tiny excerpt of the popular IMDB movie database. It comprised the tables*

- Episodes containing tv episodes belonging to different tv shows,
- Keywords saving keywords associated with those shows, and
- Locations storing places where the plots of the given tv shows play,

see Figure (17.1) on Page (174).

As seen before, every tuple t of Figure (17.1) on Page (174) is augmented by an atomic tuple event e_t and its probability $\mathbf{P}(e_t)$. Since we deal with a TID database, all given atomic tuple events are assumed to be independent from each other.

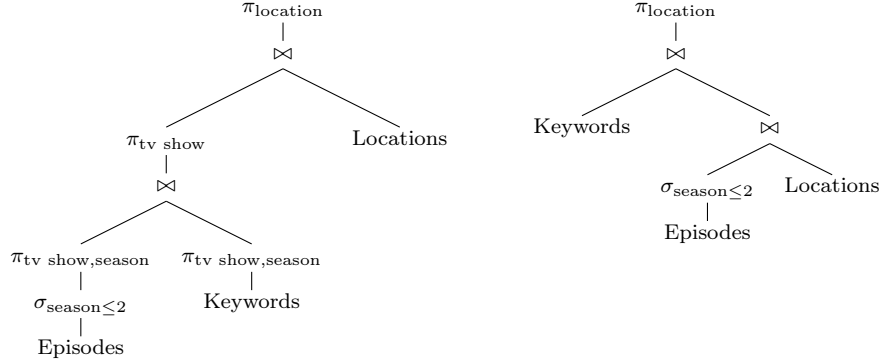
In contrast to Example (9.1) on Page (64), we study two semantically equivalent example queries, namely Q_L and Q_R shown in Figure (17.2) on Page (174). They are evaluated on the probabilistic database specified by our three IMDB tables. Both queries ask for locations where

Episodes					
	tv show	season	episode	E	P (e_t)
t_1	Dexter	1	1	e_1	0.3
t_2	Dexter	1	2	e_2	0.2
t_3	Dexter	3	3	e_3	0.4
t_4	Sopranos	2	4	e_4	0.5
t_5	Californication	2	2	e_5	0.1
t_6	Californication	2	3	e_6	0.3

Keywords					
	tv show	season	keyword	E	P (e_t)
t_7	Dexter	2	double life	e_7	0.2
t_8	Dexter	3	criminal	e_8	0.4
t_9	Sopranos	2	mob	e_9	0.5
t_{10}	Californication	2	writer	e_{10}	0.6

Locations				
	tv show	location	E	P (e_t)
t_{11}	Sopranos	USA	e_{11}	0.1
t_{12}	Sopranos	Italy	e_{12}	0.3
t_{13}	Californication	USA	e_{13}	0.8

Figure 17.1: IMDB tables with atomic tuple events

Figure 17.2: Query plans Q_L and Q_R

the story of at least one season of a TV show with at least one keyword has taken place. The considered episodes have to be part of the first two seasons.

Safe plans

In Chapter (6), we repeated various syntactic normal forms and transformations that can be applied on lineage formulas. In particular, we explained how syntactic forms can dramatically influence the costs of calculating lineage probabilities.

For example, lineage formulas written in 1OF are evaluable on TID/BID databases in linear time (Lemma (6.3) on Page (40)).

Obviously, we are interested in dealing with lineage formulas that are given in forms, which are easy to evaluate.

Ideally, we would *directly* construct a lineage formula in a tractable syntactic form *without* needing an additional transformation step into such a form. This goal can be reached, if we take care of the structure of the given input query. Please remember that the algebra operator orderings directly determine the structures of our lineage formulas, when we employ the classical construction rules of Lemma (6.1) on Page (35), or our vertical construction algorithm.

The structures of the constructed lineage formulas can be controlled by rewriting the given input query.

Dalvi, Suciu, and Ré devised in their landmark studies [24, 91, 25] two prominent algorithms that generate this type of query plans called safe plans. Please note that safe plans were originally developed for the MystiQ system [94]. In contrast to other approaches, MystiQ does not need to explicitly construct lineage formulas. It rather pushes the entire probability computation into its relational database layer. Consequently, a safe plan is originally evaluated on-the-fly within the used RDBMS. However, for comparison reasons, we study the lineage formulas explicitly constructed by safe plans.

Safe plans consist of certain operator sequences and patterns that imply tractable lineage structures, which are easy to evaluate.

Example 17.2 (Tractable lineage formulas constructed by safe plans (MystiQ)). *The example query Q_L of Figure (17.2) on Page (174) is a proper example for a safe plan. When we apply the classical construction rules on Q_L , we obtain two lineage formulas*

$$\varphi_{Q_L}^{USA} = ((e_4 \wedge e_9) \wedge e_{11}) \odot (((e_5 \odot e_6) \wedge e_{10}) \wedge e_{13})$$

and

$$\varphi_{Q_L}^{Italy} = (e_4 \wedge e_9) \wedge e_{12}$$

already studied in Example (8.3) on Page (56). Obviously, both lineage formulas are directly built in 1OF. Hence, we can calculate the probabilities $\mathbf{P}(\varphi_{Q_L}^{USA})$ and $\mathbf{P}(\varphi_{Q_L}^{Italy})$ in a straightforward way by exploiting Lemma (6.3) on Page (40):

$$\begin{aligned} \mathbf{P}(\varphi_{Q_L}^{USA}) &= \mathbf{P}(((e_4 \wedge e_9) \wedge e_{11}) \odot (((e_5 \odot e_6) \wedge e_{10}) \wedge e_{13})) \\ &= (1 - (1 - ((\mathbf{P}(e_4) * \mathbf{P}(e_9)) * \mathbf{P}(e_{11})))) * \\ &\quad (1 - ((1 - (1 - \mathbf{P}(e_5)) * (1 - \mathbf{P}(e_6)))) * \mathbf{P}(e_{10})) * \mathbf{P}(e_{13})) \\ &= (1 - (1 - ((0.5 * 0.5) * 0.1))) * (1 - ((1 - (1 - 0.1) * (1 - 0.3)) * 0.6) * 0.8) \\ &= 0.1980625 \\ \mathbf{P}(\varphi_{Q_L}^{Italy}) &= \mathbf{P}((e_4 \wedge e_9) \wedge e_{12}) \\ &= (\mathbf{P}(e_4) * \mathbf{P}(e_9)) * \mathbf{P}(e_{12}) \\ &= (0.5 * 0.5) * 0.3 \\ &= 0.075. \end{aligned}$$

Importantly, there is no additional transformation step, e.g., Shannon expansion (Section (6.3)), necessary.

Fulfillment of design goals for lineage optimization			
	design goal	safe plans (MystiQ)	lineage transformation (MayBMS/SPROUT)
1	full relational algebra support	no	no
2	unconstrained relational optimizers	no	yes
3	direct probability computation for tractable queries	yes	no

Figure 17.3: Fulfillment of design goals for lineage optimization

At first glance, safe plans seem to be the perfect way for evaluating a query on TID/BID databases, since they are capable of computing answer probabilities in \mathcal{P} -time. Unfortunately, safe plans suffer from two serious drawbacks.

There are obviously queries which cannot be rewritten into a safe plan. Otherwise, the query evaluation problem for probabilistic databases would not be $\#\mathcal{P}$ -hard [24].

In addition, Olteanu, Huang and Koch proved that even queries with a safe plan have a further downside, which they have extensively analyzed in their remarkable work [82]. In order to demonstrate this disadvantage, it has to be reminded that all safe plans enforce specific operation patterns within their query structures. Those operation patterns guarantee the desired tractable forms of the constructed lineage formulas. For instance, join and projection operations in particular are required to be performed in fixed sequences.

Strict operation orderings very often contradict the optimization strategies of the underlying RDBMS. That generally leads to a very inefficient query processing within the relational database layer [82].

Example 17.3 (Inefficient safe plans (MystiQ)). *To give an example of the conflict between relational processing and implied lineage structures, we investigate our safe plan Q_L from the view point of a relational optimizer. By doing so, we can see that the first join between the projected tables Episodes and Keywords creates unnecessary intermediate tuples (e.g., the joined tuple between t_1 and t_8). It could be easily avoided, if we would exploit the fact that the join between Episodes and Locations is more selective. Moreover, Q_L demands three expensive projection operations to ensure lineage formulas in 1OF. Olteanu et al. showed in [82] that projections are very costly operations in a safe plan.*

The negative implications of safe plans were first described in [11] and [82]. They are additionally confirmed by our experiments.

Lineage optimization within a probabilistic query engine

In order to overcome the problems of safe plans, Olteanu, Huang, and Koch proposed in [82] to split the overall query evaluation process into two parts. First, their MayBMS system executes a fast relational query plan without using fixed join orderings and unnecessary expensive projection

operations. As outcome of the relational database layer, MayBMS produces a set of lineage formulas in DNF.

Example 17.4 (Relational query plan in MayBMS). *Our example query Q_R embodies a query plan as performed in MayBMS. In contrast to $\varphi_{Q_L}^{USA}$ of Example (17.2) on Page (174), the lineage formula*

$$\varphi_{Q_R}^{USA} = (e_9 \wedge (e_4 \wedge e_{11})) \oslash (e_{10} \wedge (e_5 \wedge e_{13})) \oslash (e_{10} \wedge (e_6 \wedge e_{13}))$$

constructed for Q_R cannot be evaluated directly by Lemma (6.3) on Page (40). The subformulas of $\varphi_{Q_R}^{USA}$ are neither in 1OF nor mutually exclusive.

In the second processing phase of [82], Olteanu et. al loaded all built lineage formulas from their relational database layer (MayBMS) into their second-storage query engine (SPROUT). There, their lineage formulas are transformed and evaluated by means of a special on-the-fly algorithm. It extensively exploits the logical law of distributivity to transfer lineage formulas from DNF into 1OF. This transformation is directly combined with the probability computation of all lineage formulas.

To perform the transformation from DNF to 1OF, the lineage optimization and evaluation algorithm of SPROUT requires several expensive sorting runs over different lists of conjuncts.

The results of our experiments clearly show that the involved sorting operations can cause very ineffective probability computations, see Section (17.3).

Another negative point of SPROUT is its very limited class of supported queries.

Only tractable queries from **nrSPJ** on TID databases can be processed with the combination of MayBMS and SPROUT.

Accordingly, SPROUT does not facilitate, in contrast to BID databases, a direct modeling of attribute uncertainty, see Section (4.2).

The matrix of Figure (17.3) on Page (176) summarizes the three design goals for our optimization approach. They are derived from the advantages and disadvantages of the two discussed techniques:

- Design goal (1): full relational algebra support,
- Design goal (2): unconstrained relational optimizers, and
- Design goal (3): direct probability computation without additional lineage transformation for tractable queries.

17.2 Orthogonal combination of lineage optimizations by decoupling

In the following chapter, we propose a novel optimization technique that considerably enhances our vertical lineage construction method introduced in the last chapter. It directly exploits our core idea of decoupling the overall lineage structures from the underlying relevant domains. In such way, we can optimize the input parts for our construction algorithm independently from each other.

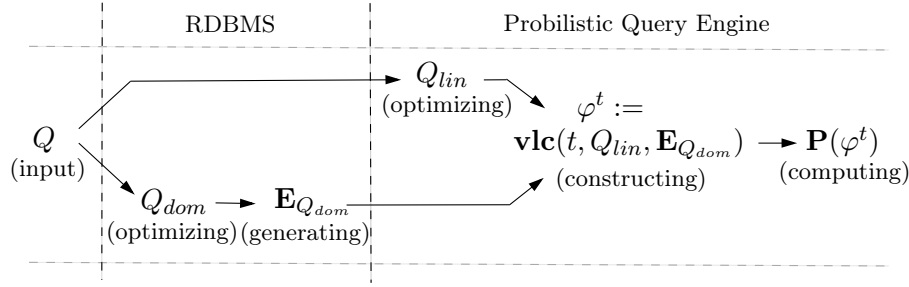


Figure 17.4: Data flow of decoupled lineage optimization

It is a well-known fact that efficient relational query plans do not imply tractable lineage structures [82]. Therefore, the already built lineage formulas have to be transformed within the probabilistic query engine via an additional optimization step. Alternatively, we could accept a poor relational query processing and compute answer probabilities directly within the RDBMS.

Our optimization approach *completely* overcomes the conflict between query plans optimized for an RDBMS and/or a probabilistic query engine.

We also start with executing an optimized relational query plan Q_{dom} without any constraints. Subsequently, the overall lineage structures in the form of a further query plan Q_{lin} get optimized *before* any lineage formulas are actually constructed within the probabilistic query engine. This is possible, because we do not encode lineage formulas *explicitly* within the relational database layer. All lineage formulas are represented by our event relation \mathbf{E}_Q and our implicitly given transformation mapping between relational algebra, relevant conditions, and lineage formulas of Figure (10.1) on Page (80).

To be more specific, we illustrate in Figure (17.4) on Page (178) the main idea of our adjusted query evaluation. First of all, our underlying RDBMS generates the event relation $\mathbf{E}_{Q_{dom}}$ based on the query plan Q_{dom} , which is specifically optimized for this task. It is not influenced by any lineage *structure* considerations.

Next, the event relation $\mathbf{E}_{Q_{dom}}$ and the original input query Q are given as inputs into our probabilistic query engine. At this point, we still deal with two specific input parts. This enables us to enhance the overall lineage structures indirectly determined by Q before our **vlc**-algorithm starts to work.

For this purpose, we rewrite the original query Q into a query plan Q_{lin} , which assures tractable lineage formulas, if tractable forms are possible in principle. To determine the query plan Q_{lin} , we can take advantage of already existing plan generating algorithms, e.g., see Dalvi, Suciu, and Re [24, 91, 25]. As a consequence, our lineage formulas are *always* built in an optimized form independently from the query plan performed within the relational database layer.

In essence, we are no longer forced to choose between an optimal plan for the underlying RDBMS (e.g., lazy plans from [82]), an optimal plan for probability computation (e.g., safe plans from [24]), or a compromise between RDBMS and probability computation (e.g., hybrid plans from [82]). Instead, we now have the option to combine the optimum on *both* sides.

Example 17.5 (Decoupled lineage optimization). *In order to demonstrate our decoupled lineage optimization, we revisit the lineage formulas built in Example (15.5) on Page (153). More concretely, we intend to reconstruct $\varphi_{\bowtie}^{(1)}, \dots, \varphi_{\setminus}^{(2)}$ already known from Figure (15.4), (15.5), and (15.6) on Page (154), (155), and (156). In contrast to the first versions generated in the last chapter, we now build them in 1OF.*

According to our concept of decoupled optimization shown in Figure (17.4) on Page (178), we first look for equivalent query plans $Q_{\bowtie, dom}, Q_{\cup, dom}$ and $Q_{\setminus, dom}$, which ensure a rapid relational query processing. For the sake of convenience, we simply use our original queries Q_{\bowtie}, Q_{\cup} and Q_{\setminus} , i.e.,

$$Q_{\bowtie, dom} := Q_{\bowtie}, \quad Q_{\cup, dom} := Q_{\cup} \quad \text{and} \quad Q_{\setminus, dom} := Q_{\setminus}.$$

Then, the relational database layer produces the known event relations of Figure (13.2) on Page (129):

$$\mathbf{E}_{Q_{\bowtie, dom}} := \mathbf{E}_{Q_{\bowtie}}, \quad \mathbf{E}_{Q_{\cup, dom}} := \mathbf{E}_{Q_{\cup}} \quad \text{and} \quad \mathbf{E}_{Q_{\setminus, dom}} := \mathbf{E}_{Q_{\setminus}}.$$

Second, we rewrite the given input queries into query plans that induce tractable lineage formulas. Let us assume that a respective optimizer delivers us the following query plans:

$$\begin{aligned} Q_{\bowtie, lin} &:= R_1 \bowtie \pi_A(R_2 \bowtie R_3) \\ Q_{\cup, lin} &:= \pi_A(\sigma_{B=4}(R_2)) \cup (R_1 \bowtie \pi_A(\sigma_{B \neq 4}(R_2))) \\ Q_{\setminus, lin} &:= R_1 \bowtie \pi_A(\sigma_{A \neq 1}(R_3)). \end{aligned}$$

All optimized plans are apparently equivalent to our original input queries from Example (5.1) on Page (30), i.e.,

$$Q_{\bowtie, lin} \equiv Q_{\bowtie}, \quad Q_{\cup, lin} \equiv Q_{\cup} \quad \text{and} \quad Q_{\setminus, lin} \equiv Q_{\setminus}.$$

With all specialized query plans in place, we again apply our vertical lineage construction algorithm in order to set up our final lineage formulas. The resulting formula trees are shown in Figure (17.5), (17.6), and (17.7) on Page (180), (181) and (182).

In comparison to the equivalent lineage formulas of the last chapter, all formula trees are now given in 1OF:

$$\begin{aligned} \varphi_{\bowtie}^{(1)} &= \underbrace{((e_1 \wedge e_3) \wedge (e_1 \wedge e_5)) \oslash ((e_1 \wedge e_4) \wedge (e_1 \wedge e_6))}_{\text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\bowtie}}, Q_{\bowtie})} \equiv \underbrace{e_1 \wedge ((e_3 \wedge e_5) \oslash (e_4 \wedge e_6))}_{\text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\bowtie, dom}}, Q_{\bowtie, lin})} \\ \varphi_{\cup}^{(1)} &= \underbrace{e_4 \vee ((e_1 \wedge e_3) \oslash (e_1 \wedge e_4))}_{\text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\cup}}, Q_{\cup})} \equiv \underbrace{e_4 \vee (e_1 \wedge e_3)}_{\text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\cup, dom}}, Q_{\cup, lin})} \\ \varphi_{\setminus}^{(1)} &\equiv \underbrace{((e_1 \wedge e_5) \oslash (e_1 \wedge e_6)) \wedge \neg(e_5 \oslash e_6)}_{\text{vlc}((1)_{x_A}, Q_{\setminus}, \mathbf{E}_{Q_{\setminus}})} \equiv \underbrace{F}_{\text{vlc}((1)_{x_A}, \mathbf{E}_{Q_{\setminus, dom}}, Q_{\setminus, lin})} \\ \varphi_{\setminus}^{(2)} &\equiv \underbrace{((e_2 \wedge e_7) \oslash (e_2 \wedge e_8))}_{\text{vlc}((2)_{x_A}, \mathbf{E}_{Q_{\setminus}}, Q_{\setminus})} \equiv \underbrace{e_2 \wedge (e_7 \oslash e_8)}_{\text{vlc}((2)_{x_A}, \mathbf{E}_{Q_{\setminus, dom}}, Q_{\setminus, lin})}. \end{aligned}$$

As a result, all new lineage formulas can now be directly evaluated without an additional transformation step on the lineage formula level.

Interestingly, the theoretical proof of our optimization concept, in contrast to its practical implementation, requires a remarkable effort. It is presented in Chapter (18) as an advanced topic. Generally speaking, it is not surprisingly that two equivalent queries applied on the same identical relevant domain lead to equivalent lineage formulas. The more challenging part is to verify the equivalent application of two sets of relevant domains contained in \mathbf{E}_Q and $\mathbf{E}_{Q_{dom}}$.

In Chapter (18), we prove that *all* equivalent query plans can be orthogonally combined in our vertical lineage construction algorithm without any further constraints. This gives us the greatest possible flexibility.

Theorem 17.1 (Decoupled lineage optimization). *Let Q be an algebra query. If we generate two equivalent algebra plans Q_{lin} and Q_{dom} , then*

$$(Q \equiv Q_{lin} \equiv Q_{dom}) \Rightarrow (\forall t : \mathbf{P}(t \in Q) = \mathbf{P}(\mathbf{vlc}(t, \mathbf{E}_{Q_{dom}}, Q_{lin})))$$

holds.

Proof. See Chapter (18). □

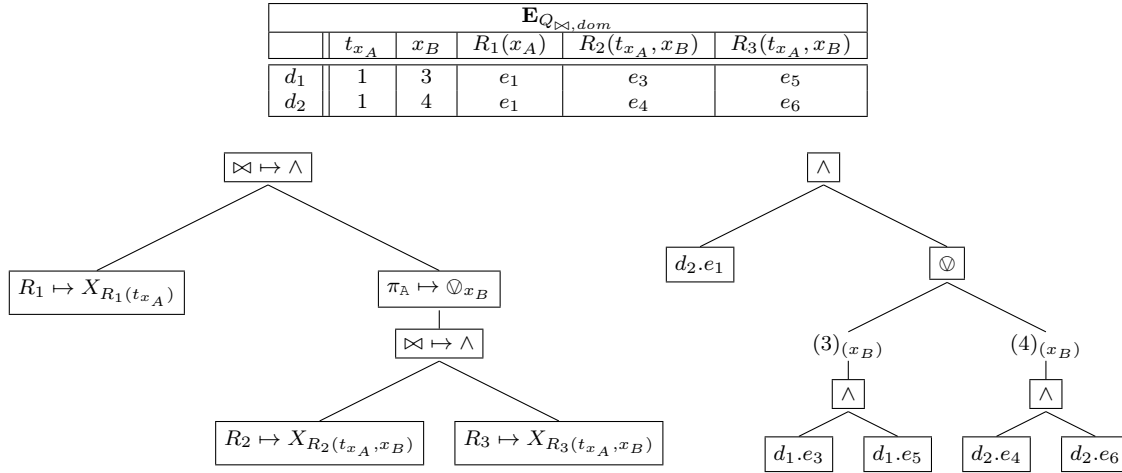


Figure 17.5: Event relation $\mathbf{E}_{Q_{\bowtie, dom}}$, the implicitly given mapping based on $Q_{\bowtie, lin}$ and the constructed formula tree $\mathbf{vlc}((1)_{x_A}, Q_{\bowtie, lin}, \mathbf{E}_{Q_{\bowtie, dom}})$

17.3 Experiments: lineage optimization and probability computation

In the following, we report the results of our third series of experiments. Please be aware that we generally distinguish between

- an optimization of the logical structure of a lineage formula and
- an optimization of the data structure representing a lineage formula.

The latter case called *lineage compression* is intensively discussed in Chapter (19). Here, we specifically address the optimization and probability computation of lineage formulas.

In this series of experiments, we again explored our five basic approaches of Section (13.5). We discuss the following topics in more detail:

- some particularities of the five basic approaches with regards to the lineage optimization and probability computation,
- the data we measured, and
- our experimental observations and conclusions.

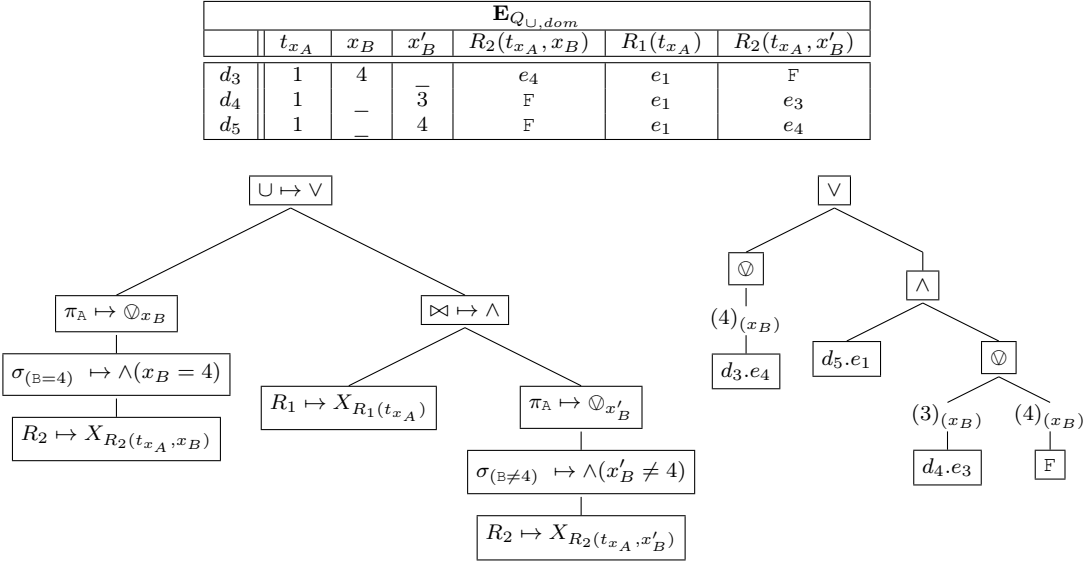


Figure 17.6: Event relation $\mathbf{E}_{Q_{\cup, dom}}$, the implicitly given transformation mapping based on $Q_{\cup, lin}$ and the constructed formula tree $\mathbf{vlc}((1)_{x_A}, Q_{\cup, lin}, \mathbf{E}_{Q_{\cup, dom}})$

Lineage optimization and probability computation: safe plans

Safe plans as employed by MystiQ [94] push the entire computation of all answer probabilities into the relational database layer. An additional construction, optimization, and evaluation of lineage formulas within a probabilistic query engine is therefore not necessary any more.

Lineage optimization and probability computation: lineage formulas in DNF

We also studied the performance of the MayBMS/SPROUT algorithm proposed by Olteanu, Huang and Koch [82]. The key idea of MayBMS/SPROUT is to generate all required lineage formulas in DNF within its relational database layer. Subsequently, the probabilistic query engine SPROUT rewrites all constructed lineage formulas into 1OF by applying the logical law of distributivity.

Example 17.6 (Distributivity applied on lineage formulas). *When we consider the following lineage formula in DNF:*

$$\varphi^t = (e_1 \odot e_2 \odot e_4) \odot (e_1 \odot e_2 \odot e_5) \odot (e_1 \odot e_3 \odot e_6) \odot (e_1 \odot e_3 \odot e_7),$$

we can repeatedly apply the logical law of distributivity on e_1, e_2 and e_3 :

$$(e_i \wedge e_j) \vee (e_i \wedge e_k) \equiv e_i \wedge (e_j \vee e_k).$$

As result, the lineage formula φ^t is folded into 1OF:

$$\begin{aligned} \varphi^t &= (e_1 \odot e_2 \odot e_4) \odot (e_1 \odot e_2 \odot e_5) \odot (e_1 \odot e_3 \odot e_6) \odot (e_1 \odot e_3 \odot e_7) \\ &\equiv e_1 \wedge ((e_2 \wedge e_4) \vee (e_2 \wedge e_5) \vee (e_3 \wedge e_6) \vee (e_3 \wedge e_7)) \\ &\equiv e_1 \wedge ((e_2 \wedge (e_4 \vee e_5)) \vee (e_3 \wedge (e_6 \vee e_7))). \end{aligned}$$

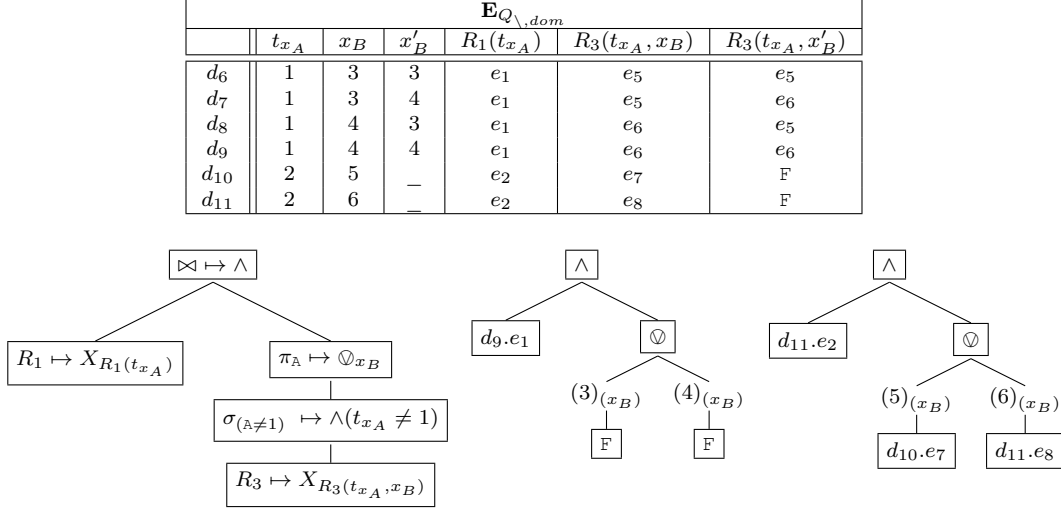


Figure 17.7: Event relation $\mathbf{E}_{Q_{\setminus, dom}}$, the implicitly given transformation mapping based on $Q_{\setminus, lin}$ and the constructed formula trees for $\mathbf{vlc}((1)_{x_A}, Q_{\setminus, lin}, \mathbf{E}_{Q_{\setminus, dom}})$ and $\mathbf{vlc}((2)_{x_A}, Q_{\setminus, lin}, \mathbf{E}_{Q_{\setminus, dom}})$

In SPROUT, the folding of conjuncts is directly merged with the computation of the final answer probabilities. The combined optimization and computation steps generally require *several* expensive sorting and re-sorting runs over the initial list of conjuncts.

Since the SPROUT algorithm only works for tractable **SPJ**-queries applied on TID databases, we could only evaluate the queries IMDB-Q1, ..., IMDB-Q3 and TPCH-Q1, ..., TPCH-Q4. The remaining queries are not supported by the SPROUT algorithm, since they also involve union and/or difference operations.

Lineage optimization and probability computation: nested lineage formulas

If a system does not favour a specific lineage optimization method, we omit an additional optimization step by constructing directly tractable lineage formulas based on the safe forms of our tested queries, see Figure (A.5) and (A.6) on Page (250) and (251). The tractability of all built lineage formulas led to very low probability computation times for all test queries, see Lemma (6.3) on Page (40).

Lineage optimization and probability computation: factor networks

Similarly to the former case, we exploited the safe forms of our tested queries (Figure (A.5) and (A.6) on Page (250) and (251)) in order to create factor networks that could be directly evaluated by Lemma (6.3) on Page (40) without any further transformation steps.

Nevertheless, it has to be stressed that the PrDB and Trio system also developed very sophisticated evaluation techniques for computing answer probabilities of lineage formulas, which are not directly given in tractable form. Such methods mainly address and improve the evaluation of hard queries. For our test case of tractable queries, we assumed that they are not significantly better than our linear-time method specified in Lemma (6.3) on Page (40).

Lineage optimization and probability computation: vertical lineage construction

Our decoupled optimization technique has been introduced in the first sections of this chapter. It basically takes advantage of two query plans Q_{dom} and Q_{lin} , which ensure an efficient relational query processing (i.e., no pre-fixed join orderings and only one final projection) and a probability computation (i.e., safe forms from Figure (A.5) and (A.6) on Page (250) and (251)) without any additional optimization steps.

Measured properties: probability computation						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	RDBMS relation(s)	1	1	1	13	7
	RDBMS tuples	5,473	91,406	5,473	369,212	5,480
	lineage nodes	0	548,436	372,678	104,212	372,678
	opt/prob steps	0	3,098,613	134,223	90,464	134,223
IMDB-Q2	RDBMS relation(s)	1	1	1	7	5
	RDBMS tuples	30	106,033	30	37,298	3,400
	lineage nodes	0	424,132	408,280	35,298	408,280
	opt/prob steps	0	3,981,635	102,175	30,055	102,175
IMDB-Q3	RDBMS relation(s)	1	1	1	13	5
	RDBMS tuples	1,516	940,008	1,516	215,636	1,520
	lineage nodes	0	3,760,032	296,748	80,636	296,748
	opt/prob steps	0	40,928,067	81,009	73,429	81,009
IMDB-Q4	RDBMS relation(s)	1	-	1	10	1
	RDBMS tuples	2,216	-	2,216	926,081	2,995
	lineage nodes	0	-	38,572	26,382	38,572
	opt/prob steps	0	-	14,075	9,643	14,075
IMDB-Q5	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	5,535	-	5,535	1,744,059	6,257
	lineage nodes	0	-	139,188	98,151	139,188
	opt/prob steps	0	-	52,189	37,916	52,189
IMDB-Q6	RDBMS relation(s)	1	-	1	12	1
	RDBMS tuples	141	-	141	655,243	1,480
	lineage nodes	0	-	5,518	4,356	5,518
	opt/prob steps	0	-	2,173	1,489	2,173

Measured computation times: probability computation						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
IMDB-Q1	relational	10.36	1.612	16.806	3.301	1.385
	lineage	0.0	1.659	0.707	2.337	1.56
	opt/prob	0.0	1.792	0.122	0.081	0.122
	total	10.36	5.063	17.635	5.719	3.067
IMDB-Q2	relational	1.109	0.428	17.971	0.368	0.267
	lineage	0.0	1.504	0.532	0.175	0.2
	opt/prob	0.0	0.658	0.09	0.005	0.089
	total	1.109	2.59	18.593	0.548	0.556
IMDB-Q3	relational	33.898	3.958	190.858	7.075	2.839
	lineage	0.0	8.249	0.359	9.289	0.297
	opt/prob	0.0	7.999	0.074	0.023	0.074
	total	33.898	20.206	191.291	16.387	3.21
IMDB-Q4	relational	13.31	-	14.77	3.699	0.097
	lineage	0.0	-	0.048	4.834	0.336
	opt/prob	0.0	-	0.016	0.011	0.016
	total	13.31	-	14.833	8.545	0.449
IMDB-Q5	relational	32.182	-	52.796	15.478	0.593
	lineage	0.0	-	0.201	64.724	1.578
	opt/prob	0.0	-	0.043	0.029	0.044
	total	32.182	-	53.041	80.23	2.215
IMDB-Q6	relational	14.48	-	19.746	5.375	0.455
	lineage	0.0	-	0.008	4.227	0.037
	opt/prob	0.0	-	0.002	0.002	0.002
	total	14.48	-	19.757	9.605	0.493

Figure 17.8: IMDB queries: measured properties and computation times for lineage optimization and probability computation

Measured data

The results obtained for the third group of our experiments are characterized by

- the accumulated number of all steps performed for optimizing and evaluating lineage formulas per approach, see the rows labeled with *opt/prob steps* in Figure (17.8) and (17.9) on Page (183) and (185), and
- the mean of the processing times (wall-clock) of 100 runs for optimizing and evaluating all lineage formula within our probabilistic query engine per approach, see the rows labeled with *opt/prob* in Figure (17.8) and (17.9) on Page (183) and (185).

Again, we also repeat and sum up the measured properties and computation times for all former processing parts.

Experimental observations

The following list discusses properties and computation times we obtained in our third series of experiments.

Measured properties: lineage optimization and probability computation steps within the probabilistic query engine

- **safe plans:** All necessary optimizations were already conducted on the query level without involving a probabilistic query engine.
- **DNF lineage:** Since lineage formulas in DNF could not be evaluated directly, they had to be optimized on the lineage formula level. The used SPROUT algorithm rewrote and evaluated all lineage formulas simultaneously executing several runs of additional sorting steps. If the list of input conjuncts came already pre-sorted from the relational database layer, no additional sorting were needed, see TPC-H-Q1 and TPC-H-Q2.
- **nested lineage, networks, vertical:** For these three approaches, all lineage formulas were directly constructed in a tractable form. Hence, no additional lineage optimization was necessary.

Measured computation times: lineage optimization and probability computation within the probabilistic query engine

- **safe plans:** No lineage optimization and probability computation were performed within our probabilistic query engine.
- **DNF lineage:** For queries with pre-sorted list of conjuncts (i.e., TPC-H-Q1 and TPC-H-Q2 only involving (1:m)-joins), we measured the fastest overall computation times. Remarkably, in all other cases, the additional optimizations on lineage formula level led to overall computations that were even worse than the ones measured for safe plans.
- **nested lineage, networks, vertical:** Not surprisingly, the evaluation times measured for the tractable lineage formulas were very low.

17.4 Summary

In this chapter, we laid out the second main contribution of this work, namely our decoupled optimization concept. It can be seamlessly integrated into the basic 2-tier architecture of our framework and fulfill *all* our design goals we initially stated:

- **Design goal (1): full relational algebra support:** our domain-oriented and lineage-oriented query plans Q_{dom} and Q_{lin} allow all relational operators,
- **Design goal (2): unconstrained relational optimizers:** our domain-oriented and lineage-oriented query plans Q_{dom} and Q_{lin} can be optimized independently from each other,

Measured properties: probability computation						
query	property	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	RDBMS relation(s)	1	1	1	6	1
	RDBMS tuples	125	1,725	125	154,047	1,725
	lineage nodes	0	5,175	4,718	2,014	4,718
	opt/prob steps	0	6,374	1,617	1,117	1,617
TPCH-Q2	RDBMS relation(s)	1	1	1	10	1
	RDBMS tuples	961	14,752	961	176,761	14,752
	lineage nodes	0	59,008	43,054	31,234	43,054
	opt/prob steps	0	66,133	17,116	10,601	17,116
TPCH-Q3	RDBMS relation(s)	1	1	1	12	6
	RDBMS tuples	38	890,072	38	239,647	11,453
	lineage nodes	0	4,450,360	72,488	60,904	72,488
	opt/prob steps	0	38,012,820	22,527	20,058	22,527
TPCH-Q4	RDBMS relation(s)	1	1	1	11	6
	RDBMS tuples	14	1,277,028	14	361,648	1,125
	lineage nodes	0	6,385,140	159,204	99,238	159,204
	opt/prob steps	0	61,921,543	43,210	37,232	43,210
TPCH-Q5	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	99	-	99	151,520	1,520
	lineage nodes	0	-	271,788	51,788	271,788
	opt/prob steps	0	-	68,343	50,233	68,343
TPCH-Q6	RDBMS relation(s)	1	-	1	11	6
	RDBMS tuples	63	-	63	348,128	5,045
	lineage nodes	0	-	233,362	128,974	233,362
	opt/prob steps	0	-	68,704	61,001	68,704
TPCH-Q7	RDBMS relation(s)	1	-	1	14	7
	RDBMS tuples	27	-	27	493,531	2,166
	lineage nodes	0	-	100,180	66,233	100,180
	opt/prob steps	0	-	29,483	28,456	29,483

Measured computation times: probability computation						
query	processing part	safe plans (MystiQ)	DNF lineage (MayBMS)	nested lineage (SPROUT2)	networks (PrDB)	vertical (Prophecy)
TPCH-Q1	relational	12.332	0.112	15.082	2.173	0.113
	lineage	0.0	0.015	0.036	3.119	0.144
	opt/prob	0.0	0.041	0.007	0.002	0.007
	total	12.332	0.168	15.125	5.295	0.228
TPCH-Q2	relational	8.296	0.489	10.96	2.353	0.488
	lineage	0.0	0.026	0.126	1.703	0.946
	opt/prob	0.0	0.521	0.016	0.014	0.015
	total	8.296	1.036	11.102	4.069	1.6
TPCH-Q3	relational	7.363	2.411	12.847	3.721	1.683
	lineage	0.0	8.407	0.109	1.891	0.542
	opt/prob	0.0	7.502	0.024	0.017	0.024
	total	7.363	18.320	12.98	5.63	2.25
TPCH-Q4	relational	14.536	2.649	55.117	5.88	1.207
	lineage	0.0	14.652	0.324	7.535	0.186
	opt/prob	0.0	11.021	0.045	0.01	0.045
	total	14.536	28.322	55.487	13.425	1.438
TPCH-Q5	relational	1.877	-	4.674	0.81	0.37
	lineage	0.0	-	0.438	1.009	0.113
	opt/prob	0.0	-	0.056	0.01	0.06
	total	1.877	-	5.167	1.829	0.543
TPCH-Q6	relational	5.675	-	12.307	3.65	0.898
	lineage	0.0	-	0.278	1.387	0.498
	opt/prob	0.0	-	0.054	0.019	0.054
	total	5.675	-	12.639	5.162	1.459
TPCH-Q7	relational	9.534	-	15.386	3.862	1.381
	lineage	0.0	-	0.122	1.979	0.347
	opt/prob	0.0	-	0.098	0.058	0.098
	total	9.534	-	15.606	5.899	1.789

Figure 17.9: TPC-H queries: measured properties for lineage optimization and probability computation within probabilistic query engine

i.e., an RDBMS optimizing Q_{dom} is not impacted by any indirect lineage structure constraints of Q_{lin} ,

- **Design goal (3): direct probability computation without additional lineage trans-**

formation for tractable queries: the safe form of our lineage-oriented query plan Q_{lin} assures tractable lineage formulas, if they are in principle possible.

Chapter 18

Advanced aspects of lineage optimization

In this chapter, we present the following advanced aspects concerning our decoupled lineage optimization approach:

- Section (18.1): a descriptive definition of minimal relevant domains,
- Section (18.2): the correctness poof of our decoupled optimization idea, and
- Section (18.3): the verification of so-called *exclusive worlds*, which are an essential part of the former correctness proof.

Example queries

For starters, we introduce three new example queries, which are more suited for the theoretical discussions to come.

Example 18.1 (Example queries and database for decoupled optimization). *In Chapter (18), we examine the following three queries:*

$$\begin{aligned} Q_1 &= \pi_{\emptyset}(R_1 \bowtie \rho_{(A' \leftarrow A)}(R_1)) \\ Q_2 &= \pi_{\emptyset}(R_1) \cup \pi_{\emptyset}(\rho_{(A' \leftarrow A)}(R_1)) \\ Q_3 &= \pi_{\emptyset}(R_1 \setminus \pi_A(R_2)). \end{aligned}$$

These queries are evaluated on a subdatabase of our running database from Example (4.1) on Page (24). The considered subdatabase simply consists of worlds which are built from all possible tuple combinations of

$$R_1(W^{max}) = \{\underbrace{(1)_A}_{t_1}, \underbrace{(2)}_{t_2}\} \quad \text{and} \quad R_2(W^{max}) = \{\underbrace{(1, 3)_{(A, B)}}_{t_3}, \underbrace{(1, 4)}_{t_4}\}.$$

Thus, it comprises $2^4 = 16$ different worlds:

$$W^{\emptyset} = \emptyset, \quad W^1 = \{t_1\}, \quad W^2 = \{t_2\}, \quad \dots, \quad W^{14} = \{t_1, t_2, t_3\}, \quad W^{max} = \{t_1, t_2, t_3, t_4\}.$$

In accordance with Section (10.2), we then set up our relevant conditions for Q_1, \dots, Q_3 as

$$\begin{aligned} \Phi_{1, \langle \bar{x} | \mathbf{D}_1^t \rangle}^{()} &\equiv (\exists x_A, x'_A : \phi_1^{()}(x_A, x'_A))_{\langle \bar{x} | \mathbf{D}_1^t \rangle} \equiv \bigvee_{(d_{x_A} \in \mathbf{D}_1^t)} \left(\bigvee_{(d_{x'_A} \in \mathbf{D}_1^t)} \phi_{1, \langle x_A, x'_A | d \rangle}^{()} \right) \\ \Phi_{2, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{()} &\equiv (\exists x_A, x'_A : \phi_2^{()}(x_A, x'_A))_{\langle \bar{x} | \mathbf{D}_2^t \rangle} \equiv \bigvee_{(d_{x_A} \in \mathbf{D}_2^t)} \left(\bigvee_{(d_{x'_A} \in \mathbf{D}_2^t)} \phi_{2, \langle x_A, x'_A | d \rangle}^{()} \right) \end{aligned}$$

$$\Phi_{\mathbf{3}, \langle \bar{x} | \mathbf{D}_3^t \rangle}^{\emptyset} \equiv (\exists x_A : \forall x_B : \phi_{\mathbf{3}}^{\emptyset}(x_A, x_B))_{\langle \bar{x} | \mathbf{D}_3^t \rangle} \equiv \bigotimes_{(d_{x_A} \in \mathbf{D}_3^t)} \left(\bigotimes_{(d_{x_B} \in \mathbf{D}_3^t)} \phi_{\mathbf{3}, \langle x_A, x_B | d \rangle}^{\emptyset} \right)$$

with following condition structures and relevant domains:

$$\begin{aligned} \phi_1^{\emptyset} &:= R_1(x_A) \wedge R_1(x'_A) & \mathbf{D}_1^{\emptyset} &= \{(1, 1)_{(x_A, x'_A)}, (1, 2), (2, 1), (2, 2)\} \\ \phi_2^{\emptyset} &:= R_1(x_A) \vee R_1(x'_A) & \mathbf{D}_2^{\emptyset} &= \{(1, _)_{(x_A, x'_A)}, (2, _), (_, 1), (_, 2)\} \\ \phi_3^{\emptyset} &:= R_1(x_A) \wedge \neg R_2(x_A, x_B) & \mathbf{D}_3^{\emptyset} &= \{(1, 3)_{(x_A, x_B)}, (1, 4), (2, _)\}. \end{aligned}$$

To determine the given relevant domains, we employ our rules of Lemma (11.3) on Page (106).

Lattice of worlds

Apart from new example queries, we take advantage of another concept known from linear algebra. *Lattices* are algebraic structures that satisfy certain axiomatic identities as commutativity, associativity, absorption and idempotence [13].

In our further discourse, we are specifically interested in the lattice of all possible worlds $L(\mathcal{W})$. It helps us to investigate and characterize the interrelations between the set of possible worlds \mathcal{W} of a probabilistic database in a more systematic way.

Definition 18.1 (Lattice $L(\mathcal{W}) = (\mathcal{W}, \sqcup, \sqcap)$). Let $\mathbf{pdb} = (\mathcal{W}, \mathbf{P})$ be a probabilistic database.

- Then, the lattice of all possible worlds is described by the tuple $L(\mathcal{W}) = (\mathcal{W}, \sqcup, \sqcap)$, which consists of the set of all possible worlds \mathcal{W} and two binary operations

$$\sqcup : \mathcal{W} \times \mathcal{W} \rightarrow \mathcal{W} \quad \text{and} \quad \sqcap : \mathcal{W} \times \mathcal{W} \rightarrow \mathcal{W}$$

defined as

$$\begin{aligned} W' \sqcup W'' &:= W''' \quad \text{with} \quad \forall R \in \mathcal{R} : R(W''') := R(W') \cup R(W'') \\ W' \sqcap W'' &:= W''' \quad \text{with} \quad \forall R \in \mathcal{R} : R(W''') := R(W') \cap R(W''). \end{aligned}$$

- Both introduced operations \sqcup and \sqcap obey the logical laws of commutativity, associativity, absorption, and idempotence.
- The operations \sqcup and \sqcap are also known as *meet* and *join*.

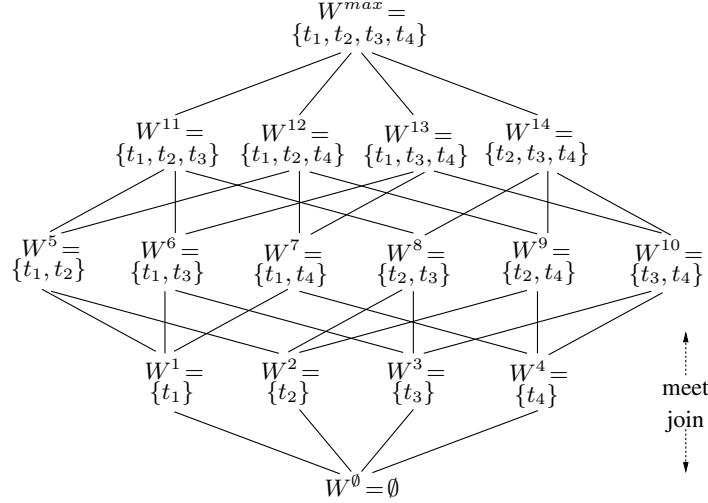
Example 18.2 (Lattice of possible worlds). In the following, we work on the lattice $L(\mathcal{W}) = (\mathcal{W}, \sqcup, \sqcap)$, which incorporates the worlds

$$\mathcal{W} = \{W^{\emptyset}, W^1, \dots, W^{14}, W^{max}\}$$

of Example (18.1) on Page (187). The lattice $L(\mathcal{W}) = (\mathcal{W}, \sqcup, \sqcap)$ is depicted in Figure (18.1) on Page (189).

18.1 Minimal relevant domains

Next, we aim to develop a more descriptive definition of a *minimal domain*. We particularly look for a criterion that determines when a domain value is *necessary* for computing the resulting

Figure 18.1: Lattice of all possible worlds $L(\mathcal{W}) = (\mathcal{W}, \sqcup, \sqcap)$

probability of an answer tuple t . Besides its theoretical meaning, we exploit in Section (18.2) this particular definition of minimal relevant domains within our main correctness proof.

Towards this goal, we first determine a minimal relevant domain $\mathbf{D}^{t, \min}$ in form of a simple elimination strategy.

Remark 18.1 (Generation of minimal domains by elimination). *From Definition (11.1) and (11.2) on Page (97) and (98), we already know our concept of satisfying worlds:*

$$\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \{W \in \mathcal{W} \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t(W)\}.$$

On the basis of satisfying worlds, we can formulate a naive method for generating a minimal relevant domain. To do so, we simply eliminate relevant domains from \mathbf{D}^t as long as the set $\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$ remains steady. When we cannot further remove domain values from \mathbf{D}^t without reducing the set $\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$, we have obviously found a minimal relevant domain.

Unfortunately, the outcome of our elimination method can depend on the order of deleting domain values from \mathbf{D}^t . Domain values can become necessary for a specific elimination order and unnecessary for another one.

Example 18.3 (Generation of minimal domains by elimination). *Let us consider the example query*

$$Q_2 = \pi_{\emptyset}(R_1) \cup \pi_{\emptyset}(\rho_{(A' \leftarrow A)}(R_1)) \equiv \{t \mid \Phi_{\mathbf{2}, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{(0)}\} = Q_2^c$$

of Example (18.1) on Page (187). We try to find a minimal subset

$$\mathbf{D}_2^{(0), \min} \quad \text{of} \quad \mathbf{D}_2^{(0)} = \{(1, _)_{(x_A, x'_A)}, (2, _), (_, 1), (_, 2)\}$$

with

$$\text{satWorlds}(\Phi_{\mathbf{2}, \langle \bar{x} | \mathbf{D}_2^{(0), \min} \rangle}^{(0)}) = \text{satWorlds}(\Phi_{\mathbf{2}, \langle \bar{x} | \mathbf{D}_2^{(0)} \rangle}^{(0)}).$$

Following Remark (18.1) on Page (189), we first fix the set of satisfying worlds $\text{satWorlds}(\Phi_{\mathbf{2}, \langle \bar{x} | \mathbf{D}_2^{(0)} \rangle}^{(0)})$, which has to be preserved during the subsequent elimination. For this

purpose, we simplify $\Phi_{2, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{()}$ by applying our Lemma (14.3) on Page (138):

$$\begin{aligned}
\Phi_{2, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{()} &\equiv \bigvee_{(d_{x_A} \in \mathbf{D}_2^t)} \left(\bigvee_{(d_{x'_A} \in \mathbf{D}_2^t)} \phi_{2, \langle x_A, x'_A | d \rangle}^{()} \right) \\
&\equiv \bigvee_{(d_{x_A} \in \mathbf{D}_2^t)} \left(\bigvee_{(d_{x'_A} \in \mathbf{D}_2^t)} (R_1(x_A) \vee R_1(x'_A))_{\langle x_A, x'_A | d \rangle} \right) \\
&\equiv (R_1(1) \vee R_1(_)) \otimes (R_1(2) \vee R_1(_)) \otimes ((R_1(_) \vee R_1(1)) \otimes (R_1(_) \vee R_1(2))) \\
&\equiv (R_1(1) \vee F) \otimes (R_1(2) \vee F) \otimes ((F \vee R_1(1)) \otimes (F \vee R_1(2))) \\
&\equiv R_1(1) \vee R_1(2).
\end{aligned}$$

Accordingly, the relevant condition $\Phi_{2, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{()}$ is satisfied in a given world W , if

$$t_1 = (1)_A \in W \quad \text{or} \quad t_2 = (2)_A \in W.$$

Using our lattice of Example (18.2) on Page (188), we can identify those worlds as follows:

$$\text{satWorlds}(\Phi_{2, \langle \bar{x} | \mathbf{D}_2^t \rangle}^{()}) = \{W^1, W^2, W^5, W^6, W^7, W^8, W^9, W^{11}, W^{12}, W^{13}, W^{14}, W^{max}\}.$$

When we exhaustively test all possible removing orders, we obtain the following minimal relevant domains:

$$\begin{aligned}
\mathbf{D}_2^{(), min_1} &= \{(1, _)_{(x_A, x'_A)}, (2, _)\} & \mathbf{D}_2^{(), min_2} &= \{(1, _)_{(x_A, x'_A)}, (_, 2)\} \\
\mathbf{D}_2^{(), min_3} &= \{(_, 1)_{(x_A, x'_A)}, (2, _)\} & \mathbf{D}_2^{(), min_4} &= \{(_, 1)_{(x_A, x'_A)}, (_, 2)\}.
\end{aligned}$$

For instance, we obtain

$$\mathbf{D}_2^{(), min_1} = \{(1, _)_{(x_A, x'_A)}, (2, _)\},$$

if we first exclude $(_, 1)_{(x_A, x'_A)}$ and $(_, 2)_{(x_A, x'_A)}$ from $\mathbf{D}_2^{()}$. In contrast, these domain values are not necessary any more, when we eliminate them at first.

Because of the ambiguities highlighted in Example (18.3) on Page (189), we change our focus from an entire relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ to its smaller parts. So, we next explore satisfying worlds for a single substituted condition structure $\text{satWorlds}(\phi_{\langle \bar{x} | d \rangle}^t)$ instead of a complete relevant condition $\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$.

Example 18.4 (Satisfying worlds for substituted condition structures). *If we substitute the condition structures $\phi_1^{()}, \dots, \phi_3^{()}$ by their relevant domains, i.e.,*

$$\begin{aligned}
\phi_1^{()} &= R_1(x_A) \wedge R_1(x'_A) & \text{with } \mathbf{D}_1^{()} &= \{(1, 1)_{(x_A, x'_A)}, (1, 2), (2, 1), (2, 2)\} \\
\phi_2^{()} &= R_1(x_A) \vee R_1(x'_A) & \text{with } \mathbf{D}_2^{()} &= \{(1, _)_{(x_A, x'_A)}, (2, _), (_, 1), (_, 2)\} \\
\phi_3^{()} &= R_1(x_A) \wedge \neg R_2(x_A, x_B) & \text{with } \mathbf{D}_3^{()} &= \{(1, 3)_{(x_A, x_B)}, (1, 4), (2, _)\},
\end{aligned}$$

we can easily read all corresponding satisfying worlds from our lattice $L(W)$ of Figure (18.1) on Page (189):

$$\begin{aligned}
Q_1 : \quad \text{satWorlds}(\phi_{1, \langle x_A, x'_A | 1, 1 \rangle}^{()}) &= \text{satWorlds}(R_1(1) \wedge R_1(1)) \\
&= \{W^1, W^5, W^6, W^7, W^{11}, W^{12}, W^{13}, W^{max}\} \\
\text{satWorlds}(\phi_{1, \langle x_A, x'_A | 1, 2 \rangle}^{()}) &= \text{satWorlds}(R_1(1) \wedge R_1(2))
\end{aligned}$$

$$\begin{aligned}
&= \{W^5, W^{11}, W^{12}, W^{max}\} \\
\text{satWorlds}(\phi_{\mathbf{1}, \langle x_A, x'_A | 2, 1 \rangle}^{()}) &= \text{satWorlds}(R_1(2) \wedge R_1(1)) \\
&= \{W^5, W^{11}, W^{12}, W^{max}\} \\
\text{satWorlds}(\phi_{\mathbf{1}, \langle x_A, x'_A | 2, 2 \rangle}^{()}) &= \text{satWorlds}(R_1(2) \wedge R_1(2)) \\
&= \{W^2, W^5, W^8, W^9, W^{11}, W^{12}, W^{14}, W^{max}\} \\
Q_2 : \quad \text{satWorlds}(\phi_{\mathbf{2}, \langle x_A, x'_A | 1, _ \rangle}^{()}) &= \text{satWorlds}(R_1(1) \vee F) \\
&= \{W^1, W^5, W^6, W^7, W^{11}, W^{12}, W^{13}, W^{max}\} \\
\text{satWorlds}(\phi_{\mathbf{2}, \langle x_A, x'_A | 2, _ \rangle}^{()}) &= \text{satWorlds}(R_1(2) \vee F) \\
&= \{W^2, W^5, W^8, W^9, W^{11}, W^{12}, W^{14}, W^{max}\} \\
\text{satWorlds}(\phi_{\mathbf{2}, \langle x_A, x'_A | _, 1 \rangle}^{()}) &= \text{satWorlds}(F \vee R_1(1)) \\
&= \{W^1, W^5, W^6, W^7, W^{11}, W^{12}, W^{13}, W^{max}\} \\
\text{satWorlds}(\phi_{\mathbf{2}, \langle x_A, x'_A | _, 2 \rangle}^{()}) &= \text{satWorlds}(F \vee R_1(2)) \\
&= \{W^2, W^5, W^8, W^9, W^{11}, W^{12}, W^{14}, W^{max}\} \\
Q_3 : \quad \text{satWorlds}(\phi_{\mathbf{3}, \langle x_A, x_B | 1, 3 \rangle}^{()}) &= \text{satWorlds}(R_1(1) \wedge \neg R_2(1, 3)) \\
&= \{W^1, W^5, W^7, W^{12}\} \\
\text{satWorlds}(\phi_{\mathbf{3}, \langle x_A, x_B | 1, 4 \rangle}^{()}) &= \text{satWorlds}(R_1(1) \wedge \neg R_2(1, 4)) \\
&= \{W^1, W^5, W^6, W^{11}\} \\
\text{satWorlds}(\phi_{\mathbf{3}, \langle x_A, x_B | 2, _ \rangle}^{()}) &= \text{satWorlds}(R_1(1) \wedge \neg F) \\
&= \{W^2, W^5, W^8, W^9, W^{11}, W^{12}, W^{14}, W^{max}\}.
\end{aligned}$$

For example, we achieve $\text{satWorlds}(\phi_{\mathbf{1}, \langle x_A, x'_A | 1, 1 \rangle}^{()})$ by collecting all worlds W where $t_1 = (1)_A$ is given in $R_1(W)$, since

$$\phi_{\mathbf{1}, \langle x_A, x'_A | 1, 1 \rangle}^{()} = R_1(1) \wedge R_1(1) \equiv R_1(1).$$

When we compare the determined worlds shown in Example (18.3) and (18.4) on Page (189) and (190), we can make three important observations.

Example 18.5 (Properties of satisfying worlds for substituted condition structures). *Let us consider Q_1 with its relevant condition and condition structure introduced in Example (18.1) and (18.4) on Page (187) and (190).*

- Then, the satisfying worlds for an entire relevant condition are covered by the satisfying worlds for its substituted condition structures, e.g.,

$$\begin{aligned}
\text{satWorlds}(\Phi_{\mathbf{1}, \langle \bar{x} | \mathbf{D}_1^t \rangle}^t) &= \bigcup_{d \in \mathbf{D}_1^t} \text{satWorlds}(\phi_{\mathbf{1}, \langle \bar{x} | d \rangle}^t) \\
&= \{W^1, W^2, W^5, W^6, W^7, W^8, W^9, W^{11}, W^{12}, W^{13}, W^{14}, W^{max}\};
\end{aligned}$$

- The satisfying worlds for different substituted condition structures are identical, e.g.,

$$\text{satWorlds}(\phi_{\mathbf{1}, \langle \bar{x} | 1, 2 \rangle}^{()}) = \{W^5, W^{11}, W^{12}, W^{max}\} = \text{satWorlds}(\phi_{\mathbf{1}, \langle \bar{x} | 2, 1 \rangle}^{()})$$

considering $(1, 2)_{(x_A, x'_A)}, (2, 1) \in \mathbf{D}_1^{()};$

- The satisfying worlds for different substituted condition structures are included in each other, e.g.,

$$\begin{aligned} \text{satWorlds}(\phi_{1, \langle \bar{x}|1,2 \rangle}^{\emptyset}) &= \{W^5, W^{11}, W^{12}, W^{max}\} \\ &\subset \{W^1, W^5, W^6, W^7, W^{11}, W^{12}, W^{13}, W^{max}\} \\ &= \text{satWorlds}(\phi_{1, \langle \bar{x}|1,1 \rangle}^{\emptyset}) \end{aligned}$$

considering $(1, 1)_{(x_A, x'_A)}, (1, 2) \in \mathbf{D}_1^{\emptyset}$.

Referring to the three exemplified properties, we further refine our description of a minimal relevant domain. Thus, a set of relevant domain values is minimal, if none of their corresponding world sets is either identical or a subset of another one. Otherwise, we can remove at least one domain value. To formalize that conclusion, we introduce a binary equivalence relation over the set of all possible worlds \mathcal{W} . It groups domain values which have identical sets of satisfying worlds.

Definition 18.2 (Equivalence relation over relevant domain values). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}^t} \phi_{\langle \bar{x}|d \rangle}^t\}.$$

- Then, we specify the binary equivalence relation $\sim : \subseteq \mathbf{D}^t \times \mathbf{D}^t$ as

$$(d \sim \hat{d}) :\Leftrightarrow (\text{satWorlds}(\phi_{Q, \langle \bar{x}|d \rangle}^t) = \text{satWorlds}(\phi_{Q, \langle \bar{x}|\hat{d} \rangle}^t)).$$

- A corresponding domain class $[d]$ is defined as

$$[d] := \{\hat{d} \in \mathbf{D}^t \mid \hat{d} \sim d\}.$$

- For the entire partition induced by \sim , we write:

$$(\mathbf{D}^t \setminus \sim) := \{[d_1], \dots, [d_n]\} \quad \text{with} \quad \mathbf{D}^t = [d_1] \dot{\cup} \dots \dot{\cup} [d_n].$$

- Moreover, we extend our syntactic substitution operators of Definition (10.3) on Page (82) in order to handle domain classes as well, i.e., $\Phi_{\langle \bar{x}|\mathbf{D}^t \rangle}^t$ replaces all variables \bar{x} within Φ^t by an arbitrary member of $[d]$.
- Our rules for deriving a propositional condition (Lemma (10.3) on Page (88)) can also be applied in combination with domain classes. The central rules resolving quantifiers are then adjusted to:

$$\begin{aligned} \Phi^t = \exists y : \Phi_1^t & : \Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim) \rangle}^t := \bigvee_{[d_y] \in (\mathbf{D}_y^t \setminus \sim)} (\Phi_{1, \langle \bar{x} | (\mathbf{D}^t \setminus \sim) \rangle}^t)_{\langle y | [d_y] \rangle} \\ \Phi^t = \forall y : \Phi_1^t & : \Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim) \rangle}^t := \bigwedge_{[d_y] \in (\mathbf{D}_y^t \setminus \sim)} (\Phi_{1, \langle \bar{x} | (\mathbf{D}^t \setminus \sim) \rangle}^t)_{\langle y | [d_y] \rangle}. \end{aligned}$$

Example 18.6 (Equivalence relation over relevant domain values). *By applying Definition (18.2) on Page (192), we set up the following (\sim) -partitions for our example queries Q_1, \dots, Q_3 :*

$$\begin{aligned} (\mathbf{D}_1^{\emptyset} \setminus \sim) &:= \{(1, 1)_{(x_A, x'_A)}, (1, 2), (2, 1), (2, 2)\} \\ (\mathbf{D}_2^{\emptyset} \setminus \sim) &:= \{(1, _)_{(x_A, x'_A)}, (_, 1), (2, _), (_, 2)\} \end{aligned}$$

$$(\mathbf{D}_3^0 \setminus \sim) := \{\{(1, 3)_{(x_A, x_B)}\}, \{(1, 4)\}, \{(2, _)\}\}.$$

Besides equality, we express the *inclusion* of satisfying world sets. We call that specific property as *covering of domain classes*.

Definition 18.3 (Covering of domain classes). *Let $[d]$ and $[\hat{d}]$ be two different domain classes of a given (\sim) -partition.*

- *We then define that*

$$([d] \text{ covers } [\hat{d}]) :\Leftrightarrow (\text{satWorlds}(\phi_{\langle \bar{x} | \hat{d} \rangle}^t) \subset \text{satWorlds}(\phi_{\langle \bar{x} | d \rangle}^t)).$$

- *Moreover, we state that*

$$([d] \text{ is uncovered}) :\Leftrightarrow (\nexists [\hat{d}] \in (\mathbf{D}_Q^t \setminus \sim) : [\hat{d}] \text{ covers } [d]).$$

- *Additionally, we specify the set of all uncovered domain classes*

$$(\mathbf{D}^t \setminus \sim)^{uc} := \{[d] \in (\mathbf{D}^t \setminus \sim) \mid [d] \text{ is uncovered}\}$$

with

$$(\mathbf{D}^t \setminus \sim)^{uc} \subseteq (\mathbf{D}^t \setminus \sim).$$

Example 18.7 (Covering of domain classes). *For the example query Q_1 , we identify two covered domain classes: $[(1, 2)_{(x_A, x'_A)}]$ and $[(2, 1)_{(x_A, x'_A)}]$. They are both covered by $[(1, 1)_{(x_A, x'_A)}]$, since*

$$\begin{aligned} \text{satWorlds}(\phi_{1, \langle x_A, x'_A | 1, 2 \rangle}^0) &\subset \text{satWorlds}(\phi_{1, \langle x_A, x'_A | 1, 1 \rangle}^0) \quad \text{and} \\ \text{satWorlds}(\phi_{1, \langle x_A, x'_A | 2, 1 \rangle}^0) &\subset \text{satWorlds}(\phi_{1, \langle x_A, x'_A | 1, 1 \rangle}^0), \end{aligned}$$

see the world sets in Example (18.4) on Page (190).

In addition, we can provide the sets of all uncovered domain classes to all our example queries Q_1, \dots, Q_3 in the following form:

$$\begin{aligned} (\mathbf{D}_1^0 \setminus \sim)^{uc} &:= \{\{(1, 1)_{(x_A, x'_A)}\}, \{(2, 2)\}\} \\ (\mathbf{D}_2^0 \setminus \sim)^{uc} &:= \{\{(1, _)_{(x_A, x'_A)}, (_, 1)\}, \{(2, _), (_, 2)\}\} \\ (\mathbf{D}_3^0 \setminus \sim)^{uc} &:= \{\{(1, 3)_{(x_A, x_B)}\}, \{(1, 4)\}, \{(2, _)\}\}. \end{aligned}$$

Taking our introduced equivalence relation into consideration, we hypothesize that all members of a covered domain class are not necessary to build a minimal domain, since they can only contribute satisfying worlds, which are also provided by their covering domain classes.

Lemma 18.1 (Necessary domain classes for computing $\mathbf{P}(t \in Q)$). *Let Q be an algebra query with its equivalent domain calculus query:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t\}.$$

Then, only uncovered domain classes are required for computing the answer probabilities $\mathbf{P}(t \in Q)$. In other words,

Property (A): $\forall [d] \in (\mathbf{D}^t \setminus \sim) : [d] \text{ is uncovered} \Rightarrow \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) \neq \mathbf{P}(t \in Q)$ and
 Property (B): $\forall [d] \in (\mathbf{D}^t \setminus \sim) : [d] \text{ is covered} \Rightarrow \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) = \mathbf{P}(t \in Q)$

hold.

Proof. We verify Property (A) and (B) by an induction proof over the number of n-ary operations \otimes/\oplus involved in $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$.

I.) Induction basis ($n = 0$ n-ary operations):

When a relevant condition $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ has no n-ary operations, the original first-order condition Φ^t does not include any quantifiers. Then, we know from the construction of Φ^t (Lemma (10.1) on Page (84)) that there are no variables in Φ^t . Consequently, the underlying relevant domain and its partition are empty, see Definition (10.12) and (18.2) on Page (91) and (192). This directly assures our Property (A) and (B).

II.) Induction assumption (n n-ary operations):

The Properties

Property (A): $\forall [d] \in (\mathbf{D}^t \setminus \sim) : [d] \text{ is uncovered} \Rightarrow \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) \neq \mathbf{P}(t \in Q)$ and
 Property (B): $\forall [d] \in (\mathbf{D}^t \setminus \sim) : [d] \text{ is covered} \Rightarrow \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) = \mathbf{P}(t \in Q)$

hold for all relevant conditions $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$ with n or less n-ary operations.

III.) Induction step ($n + 1$ n-ary operations):

Let

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \overline{\bigotimes_{d \in \mathbf{D}^t}} \phi_{\langle \bar{x} | d \rangle}^t$$

be our considered relevant condition in PNF. After splitting the variable set \bar{x} into

$$\bar{x} = \{x\} \dot{\cup} \bar{y} \quad \text{with} \quad \mathbf{D}^t = \mathbf{D}_x^t \times \mathbf{D}_{\bar{y}}^t$$

and introducing

$$\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t := \overline{\bigotimes_{d_{\bar{y}} \in \mathbf{D}_{\bar{y}}^t}} \phi_{\langle \bar{y} | d_{\bar{y}} \rangle}^t,$$

we examine two disjoint cases according to the two types of n-ary operators \otimes and \oplus .

Case (1): $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \bigotimes_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle}$

The proof of Property (A) essentially relies on a lemma discussed in all details in Section (18.3). Lemma (18.4) on Page (203) ensures that there exists a special world $W^{[d]}$ for each uncovered domain class $[d]$, which is called *exclusive world* of $[d]$. In a nutshell, only domain values from the class $[d]$ are able to satisfy the corresponding substituted condition structure $\phi_{\langle \bar{x} | [d] \rangle}^t$ in its exclusive world $W^{[d]}$, i.e.,

$$(W \text{ is exclusive world of } [d]) :\Leftrightarrow ((\phi_{\langle \bar{x} | [d] \rangle}^t(W) \equiv \mathbf{T}) \wedge (\forall [\hat{d}] \in (\mathbf{D}^t \setminus \sim)^{uc} : [\hat{d}] \neq [d] \Rightarrow \phi_{\langle \bar{x} | [\hat{d}] \rangle}^t(W) \equiv \mathbf{F})).$$

The exclusive world $W^{[d]}$ represents the proof of the necessity of its associated uncovered domain class $[d]$. Lemma (18.4) on Page (203) shows that such a world always exists for an uncovered domain class.

Property (A): To begin with, we restrict the relevant condition $\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t$ to domain values taken from uncovered domain classes, as justified by Property (B) of our induction assumption:

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \Leftrightarrow (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | \mathbf{D}_x^t \rangle} \Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle}.$$

Moreover, we generalize d_x to $[d_x]$:

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle} \Leftrightarrow \bigvee_{[d_x] \in (\mathbf{D}_x^t \setminus \sim)} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | [d_x] \rangle},$$

since all members of a domain class are satisfied in the same set of worlds, see Definition (18.3) on Page (193) .

Next, we assume a specific *uncovered* domain class $[\tilde{d}_x]$ with its exclusive worlds $W^{[\tilde{d}_x]}$. Then, we can conclude from Lemma (18.4) on Page (203) that the corresponding disjunctive operand $(\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | [\tilde{d}_x] \rangle}$ of

$$\bigvee_{[d_x] \in (\mathbf{D}_x^t \setminus \sim)} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | [d_x] \rangle}$$

is the one and only that can be fulfilled in the exclusive world $W^{[\tilde{d}_x]}$.

Using our induction assumption, we can be also sure that there is an uncovered domain class $[\tilde{d}_{\bar{y}}] \in (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc}$ with $[\tilde{d}] := [\tilde{d}_x \bullet \tilde{d}_{\bar{y}}]$. Its respective world $W^{[\tilde{d}]} = W^{[\tilde{d}_x \bullet \tilde{d}_{\bar{y}}]}$ is exclusive. That is, we can separate the underlying satisfying worlds as:

$$\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [\tilde{d}] \rangle}^t) \dot{\cup} \underbrace{\text{satWorlds}((\hat{\Phi}_{\langle \bar{y} | [\tilde{d}_{\bar{y}}] \rangle}^t)_{\langle x | [\tilde{d}_x] \rangle})}_{\{W^{[\tilde{d}]}\}}.$$

If we exploit these world sets in order to compute answer probabilities via Definition (10.10) on Page (87) , we obtain

$$\begin{aligned} \mathbf{P}(t \in Q) &= \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) \\ &= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)} \mathbf{P}(W) \\ &= \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [\tilde{d}] \rangle}^t)} \mathbf{P}(W) + \mathbf{P}(W^{[\tilde{d}]}) \\ &= \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [\tilde{d}] \rangle}^t) + \mathbf{P}(W^{[\tilde{d}]}). \end{aligned}$$

Since Definition (4.1) on Page (23) assures $\mathbf{P}(W^{[\tilde{d}]}) > 0$, our claimed proposition

$$\mathbf{P}(t \in Q) \neq \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [\tilde{d}] \rangle}^t)$$

is proven.

Property (B): To show that covered domain classes are not needed for computing $\mathbf{P}(t \in Q)$, we prove an equivalent proposition. It claims that uncovered domain classes are sufficient for determining $\mathbf{P}(t \in Q)$, i.e.,

$$\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \mathbf{P}(\Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim)^{uc} \rangle}^t).$$

To do so, we simply transfer the logical disjunction operator \bigvee of

$$\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle}$$

to its corresponding union set operation \cup , which is applied on its associated sets of satisfying worlds. That means, we determine $\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$ as a union of all satisfying worlds delivered by the disjunctive operands of $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. Additionally, we benefit from the fact that the satisfying worlds of all covered domain classes are already included in the satisfying world sets of their covering domain classes (Definition (18.3)) on Page (193):

$$\begin{aligned}
\text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) &= \text{satWorlds}\left(\bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle}\right) \\
&= \text{satWorlds}\left(\bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle}\right) \\
&= \text{satWorlds}\left(\bigvee_{[d_x] \in (\mathbf{D}_x^t \setminus \sim)} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | [d_x] \rangle}\right) \\
&= \bigcup_{[d_x] \in (\mathbf{D}_x^t \setminus \sim)} (\text{satWorlds}((\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | [d_x] \rangle})) \\
&= \bigcup_{[d] \in (\mathbf{D}^t \setminus \sim)^{uc}} \text{satWorlds}(\Phi_{\langle \bar{x} | [d] \rangle}^t) \\
&= \text{satWorlds}\left(\bigvee_{[d] \in (\mathbf{D}^t \setminus \sim)^{uc}} \Phi_{\langle \bar{x} | [d] \rangle}^t\right) \\
&= \text{satWorlds}(\Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim)^{uc} \rangle}^t).
\end{aligned}$$

Subsequently, we employ Definition (10.10) on Page (87) and obtain

$$\mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t) = \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)} \mathbf{P}(W) = \sum_{W \in \text{satWorlds}(\Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim)^{uc} \rangle}^t)} \mathbf{P}(W) = \mathbf{P}(\Phi_{\langle \bar{x} | (\mathbf{D}^t \setminus \sim)^{uc} \rangle}^t).$$

In conjunction with $\mathbf{P}(t \in Q) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t)$ assured by Lemma (11.2) on Page (99), it verifies that all uncovered domain classes are sufficient.

Case (2): $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle}$

For Case (2), our idea is the application of Lemma (14.3) on Page (138). More precisely, we intend to replace all relation predicates of the form $R(x, \bar{y}, \bar{c})$ by F within $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t$. By doing so, we reduce the total number of variables in Φ^t and can subsequently exploit our induction assumption.

To begin with, we can assume that $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \top$. Otherwise, we would achieve an empty partition $(\mathbf{D}^t \setminus \sim)$, which directly confirms our Property (A) and (B).

Because $\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t \equiv \top$, we certainly know that there is at least one satisfying world for each $d_x \in \mathbf{D}_x$ where $(\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle}$ is fulfilled. Please mind that we connect our substituted operands *conjunctively*. As a consequence, \mathbf{D}_x^t equals \mathbf{D}_x . It means that we have to range over the entire domain \mathbf{D}_x in order to address all relevant domain values of x . In Definition (11.4) on Page (104), we introduced the notation $d_x = (_)$ to indicate this kind of relevant domain values.

Lemma (14.3) on Page (138) then allows us to replace all relation predicates $R(x, \bar{y}, \bar{c}) = R(_, \bar{y}, \bar{c})$ with F. In this way, we obtain a relevant condition without an n-ary conjunction that iterates over \mathbf{D}_x^t :

$$\begin{aligned}
\Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t &\Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | \mathbf{D}_{\bar{y}}^t \rangle}^t)_{\langle x | d_x \rangle} \\
&\Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} (\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle} & \text{(IA)} \\
&\Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} ((\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle} \wedge R(_, \bar{y}, \bar{c}))_{\langle x | d_x \rangle} & (\mathbf{D}_x^t = \mathbf{D}_x) \\
&\Leftrightarrow \bigvee_{d_x \in \mathbf{D}_x^t} ((\hat{\Phi}_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}^t)_{\langle x | d_x \rangle} \wedge F)_{\langle x | d_x \rangle} & \text{(Lemma (14.3))}
\end{aligned}$$

$$\Leftrightarrow (\hat{\Phi}_{\langle R(x, \bar{y}, \bar{c}) | \mathbb{F} \rangle}^t)_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}.$$

Finally, our induction assumption applied on $(\hat{\Phi}_{\langle R(x, \bar{y}, \bar{c}) | \mathbb{F} \rangle}^t)_{\langle \bar{y} | (\mathbf{D}_{\bar{y}}^t \setminus \sim)^{uc} \rangle}$ verifies Property (A) and (B). \square

By exploiting our last lemma, we eventually obtain our descriptive definition of a minimal relevant domain.

Lemma 18.2 (Minimal relevant domain). *Let Q be an algebra query and*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} | \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \bigotimes_{d \in \mathbf{D}^t} \phi_{\langle \bar{x} | d \rangle}^t\}$$

be its equivalent domain calculus query. If $(\mathbf{D}^t \setminus \sim)^{uc}$ is the set of all uncovered domain classes of a relevant domain \mathbf{D}^t , then

$$\begin{aligned} \mathbf{D}^{t, min} \subseteq \mathbf{D}^t \text{ is minimal} : \Leftrightarrow & ((\forall [d] \in (\mathbf{D}^t \setminus \sim)^{uc} : |\mathbf{D}^{t, min} \cap [d]| = 1) \wedge \\ & (\forall [d] \notin (\mathbf{D}^t \setminus \sim)^{uc} : |\mathbf{D}^{t, min} \cap [d]| = 0)). \end{aligned}$$

Proof. Lemma (18.1) on Page (193) already proved that covered classes are obsolete and all uncovered classes are necessary. The minimal numbers of representatives per covered and uncovered class are then zero and one, respectively. \square

Example 18.8 (Minimal relevant domains). *If we combine Lemma (18.2) on Page (197) with the uncovered domain classes of Example (18.7) on Page (193), we can list the final minimal relevant domains for Q_1, \dots, Q_3 as follows:*

$$\begin{aligned} \mathbf{D}_1^{(), min} &= \{(1, 1)_{(x_A, x_B)}, (2, 2)\} \\ \mathbf{D}_2^{(), min_1} &= \{(1, _)_{(x_A, x_B)}, (2, _)\} \\ \mathbf{D}_2^{(), min_2} &= \{(1, _)_{(x_A, x_B)}, (_, 2)\} \\ \mathbf{D}_2^{(), min_3} &= \{(_, 1)_{(x_A, x_B)}, (2, _)\} \\ \mathbf{D}_2^{(), min_4} &= \{(_, 1)_{(x_A, x_B)}, (_, 2)\} \\ \mathbf{D}_3^{(), min} &= \{(1, 3)_{(x_A, x_B)}, (1, 4), (2, _)\}. \end{aligned}$$

We already exemplified in Remark (18.1) on Page (189) that minimal relevant domains are not unique, e.g., Q_2 with $\mathbf{D}_2^{(), min_1}, \dots, \mathbf{D}_2^{(), min_4}$. However, in contrast to Remark (18.1) on Page (189), we have now a descriptive characterization of necessary domain values by means of uncovered domain classes.

Before we present the final correctness proof of our optimization approach, we summarize the explanations given so far. More specifically, we show that our **vlc**-algorithm is capable of constructing equivalent lineage formulas, even if they are derived from very diverse event relations. It also illustrates where minimal relevant domains are contained in the event relations.

Example 18.9 (Diverse event relations). *In Figure (17.5), (17.6) and (17.7) on Page (180), (181), and (182), we gave examples for generating tractable lineage formulas based on optimized*

\mathbf{E}_{Q_1}				
$[d]$	x_A	x'_A	$R_1(x_A)$	$R_1(x'_A)$
$[d_1]$	1	1	e_1	e_1
$[d_2]$	1	2	e_1	e_2
$[d_2]$	2	1	e_2	e_1
$[d_3]$	2	2	e_2	e_2

$\mathbf{E}_{Q_{1,dom}}$				
$[d]$	x_A	x'_A	$R_1(x_A)$	$R_1(x'_A)$
$[d_1]$	1	1	e_1	e_1
$[d_3]$	2	2	e_2	e_2

\mathbf{E}_{Q_2}				
$[d]$	x_A	x'_A	$R_1(x_A)$	$R_1(x'_A)$
$[d_4]$	1	—	e_1	F
$[d_5]$	2	—	e_2	F
$[d_4]$	—	1	F	e_1
$[d_5]$	—	2	F	e_2

$\mathbf{E}_{Q_{2,dom}}$		
$[d]$	x_A	$R_1(x_A)$
$[d_4]$	1	e_1
$[d_5]$	2	e_2

\mathbf{E}_{Q_3}				
$[d]$	x_A	x_B	$R_1(x_A)$	$R_2(x_A, x_B)$
$[d_6]$	1	3	e_1	e_3
$[d_7]$	1	4	e_1	e_4
$[d_8]$	2	—	e_2	F

$\mathbf{E}_{Q_{3,dom}}$						
$[d]$	x_A	x_B	x'_A	$R_1(x_A)$	$R_2(x_A, x_B)$	$R_1(x'_A)$
$[d_6]$	1	3	—	e_1	e_3	F
$[d_7]$	1	4	—	e_1	e_4	F
$[d_8]$	2	—	—	e_2	F	F
$[d_6]$	1	3	1	e_1	e_3	e_1
$[d_7]$	1	4	1	e_1	e_4	e_1
$[d_8]$	2	—	2	e_2	F	e_2

Figure 18.2: Previous and new event relations for Q_1, \dots, Q_3

query plans. For this purpose, we replaced the original query plans Q_{\bowtie}, Q_{\cup} and Q_{\setminus} in Example (17.5) on Page (179) with the lineage-oriented query plans $Q_{\bowtie,lin}, Q_{\cup,lin}$, and $Q_{\setminus,lin}$. The corresponding event relations kept their original contents and structures.

Now, we consider the other way around, i.e., the lineage-oriented query plans are kept and the domain-oriented query plans are adjusted. In detail, we introduce the three domain-oriented query plans

$$\begin{aligned}
Q_{1,dom} &= \pi_{\emptyset}(\sigma_{(A=A')}(R_1 \bowtie \rho_{(A' \leftarrow A)}(R_1))) \\
Q_{2,dom} &= \pi_{\emptyset}(R_1) \\
Q_{3,dom} &= \pi_{\emptyset}((R_1 \setminus \pi_A(R_2)) \cup \pi_A(\sigma_{(A=A')}((R_1 \setminus \pi_A(R_2)) \bowtie \rho_{(A' \leftarrow A)}(R_1))))).
\end{aligned}$$

They are responsible for generating our new event relations, see Definition (13.1) on Page (124). At the same time, we still use the query plans Q_1, Q_2 and Q_3 , which determine the structures of our final lineage formulas.

In Figure (18.2) on Page (198), we compare the event relations created for Q_1, \dots, Q_3 and $Q_{1,dom}, \dots, Q_{3,dom}$. Please note that the first column of each table additionally contains the domain class $[d]$ of a specific domain value.

As predicated in Lemma (18.2) on Page (197), all uncovered domain class are represented in our constructed event relations of Figure (18.2) on Page (198):

$$\begin{aligned}
(\mathbf{D}_1^{\emptyset} \setminus \sim)^{uc} &= \underbrace{\{(1, 1)_{(x_A, x'_A)}\}}_{[d_1]} \underbrace{\{(2, 2)\}}_{[d_3]} \\
(\mathbf{D}_2^{\emptyset} \setminus \sim)^{uc} &= \underbrace{\{(1, _)_{(x_A, x'_A)}, (_, 1)\}}_{[d_4]} \underbrace{\{(2, _), (_, 2)\}}_{[d_5]} \\
(\mathbf{D}_3^{\emptyset} \setminus \sim)^{uc} &= \underbrace{\{(1, 3)_{(x_A, x_B)}\}}_{[d_6]} \underbrace{\{(1, 4)\}}_{[d_7]} \underbrace{\{(2, _)\}}_{[d_8]}.
\end{aligned}$$

Although our event relation pairs

$$\mathbf{E}_{Q_1} \text{ vs. } \mathbf{E}_{Q_{1,dom}}, \quad \mathbf{E}_{Q_2} \text{ vs. } \mathbf{E}_{Q_{2,dom}} \quad \text{and} \quad \mathbf{E}_{Q_3} \text{ vs. } \mathbf{E}_{Q_{3,dom}}$$

have very different structures and contents, we can always be sure that our algorithm produces equivalent lineage formulas:

$$\begin{aligned}
\mathbf{vlc}(\cdot, Q_1, \mathbf{E}_{Q_1}) &= (e_1 \wedge e_1) \otimes (e_1 \wedge e_2) \otimes (e_2 \wedge e_1) \otimes (e_2 \wedge e_2) \\
&\equiv e_1 \vee e_2 \\
&\equiv (e_1 \wedge e_1) \otimes (e_2 \wedge e_2) \\
&= \mathbf{vlc}(\cdot, Q_1, \mathbf{E}_{Q_{1,dom}}) \\
\mathbf{vlc}(\cdot, Q_2, \mathbf{E}_{Q_2}) &= (e_1 \otimes e_2) \vee (e_1 \otimes e_2) \\
&\equiv e_1 \vee e_2 \\
&\equiv (e_1 \otimes e_2) \vee F \\
&= \mathbf{vlc}(\cdot, Q_2, \mathbf{E}_{Q_{2,dom}}) \\
\mathbf{vlc}(\cdot, Q_3, \mathbf{E}_{Q_3}) &= (e_1 \wedge \neg(e_3 \otimes e_4)) \otimes (e_2 \wedge \neg(F)) \\
&= \mathbf{vlc}(\cdot, Q_3, \mathbf{E}_{Q_{3,dom}}).
\end{aligned}$$

In essence, a relational optimizer can manipulate query plans as $Q_{1,dom}, \dots, Q_{3,dom}$ completely independently from query plans as Q_1, \dots, Q_3 , which determine the structure of lineage formulas.

18.2 Correctness proof for decoupled optimization

In Section (16.1), we already verified that our **vlc**-algorithm correctly constructs lineage formulas, which capture the desired answer probabilities, i.e.,

$$\mathbf{P}(t \in Q) = \mathbf{P}(\mathbf{vlc}(t, Q, \mathbf{E}_Q)).$$

Our decoupled optimization method guarantees that the parameters of the **vlc**-algorithm can also be derived from two equivalent query plans Q_{lin} and Q_{dom} :

$$(Q \equiv Q_{lin} \equiv Q_{dom}) \Rightarrow (\mathbf{P}(t \in Q) = \mathbf{P}(\mathbf{vlc}(t, Q_{lin}, \mathbf{E}_{Q_{dom}})).$$

This obviously holds, if the used event relations can be considered as equivalent in regards of our algorithm, i.e.,

$$\mathbf{E}_Q \stackrel{?}{\equiv} \mathbf{E}_{Q_{lin}} \stackrel{?}{\equiv} \mathbf{E}_{Q_{dom}}.$$

In other words, we can interchangeably use Q, Q_{lin}, Q_{dom} and $\mathbf{E}_Q, \mathbf{E}_{Q_{lin}}, \mathbf{E}_{Q_{dom}}$ as parameters for the **vlc**-algorithm, if we prove the case of equivalent event relations.

Definition 18.4 (Equivalence of event relations). *Let Q and \hat{Q} be two algebra queries with their equivalent domain calculus queries:*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D}^t \rangle}^t\} \equiv \{t \mid \overline{\bigotimes \bigotimes}_{d \in \mathbf{D}^t} \phi_{\langle \bar{x} \mid d \rangle}^t\}$$

and

$$\hat{Q}^c = \{t \mid \hat{\Phi}_{\langle \bar{x} \mid \hat{\mathbf{D}}^t \rangle}^t\} \equiv \{\overline{\bigotimes \bigotimes}_{d \in \hat{\mathbf{D}}^t} \hat{\phi}_{\langle \bar{x} \mid d \rangle}^t\}.$$

Then, the event relations \mathbf{E}_Q and $\mathbf{E}_{\hat{Q}}$ are said to be equivalent, if and only if

- their extracted sets of all possible answers are identical:

$$Q_{\text{poss}(\mathcal{W})} = \hat{Q}_{\text{poss}(\mathcal{W})} \quad \text{and}$$

- their extracted sets of uncovered domain classes are identical:

$$\forall t \in Q_{\text{poss}(\mathcal{W})} : ((\mathbf{D}^t \setminus \sim)^{uc} = (\hat{\mathbf{D}}^t \setminus \sim)^{uc}).$$

Example 18.10 (Equivalence of event relations). *In Example (18.9) on Page (197) and Figure (18.2) on Page (198), we presented three pairs of equivalent event relations:*

$$\mathbf{E}_{Q_1}^{()} \equiv \mathbf{E}_{Q_1, dom}^{()}, \quad \mathbf{E}_{Q_2}^{()} \equiv \mathbf{E}_{Q_2, dom}^{()} \quad \text{and} \quad \mathbf{E}_{Q_3}^{()} \equiv \mathbf{E}_{Q_3, dom}^{()}.$$

Their equivalences is assured by

- *their extracted sets of all possible answers*

$$Q_{1, poss(\mathcal{W})} = \dots = Q_{3, poss(\mathcal{W})} = Q_{1, dom, poss(\mathcal{W})} = \dots = Q_{3, dom, poss(\mathcal{W})} = \{()\} \quad \text{and}$$

- *their extracted sets of uncovered domain classes*

$$\begin{aligned} (\mathbf{D}_1^{()} \setminus \sim)^{uc} &= \{[d_1], [d_3]\} = (\mathbf{D}_{1, dom}^{()} \setminus \sim)^{uc} \\ (\mathbf{D}_2^{()} \setminus \sim)^{uc} &= \{[d_4], [d_5]\} = (\mathbf{D}_{2, dom}^{()} \setminus \sim)^{uc} \\ (\mathbf{D}_3^{()} \setminus \sim)^{uc} &= \{[d_6], [d_7], [d_8]\} = (\mathbf{D}_{3, dom}^{()} \setminus \sim)^{uc}. \end{aligned}$$

Thanks to construction rules of Definition (13.1) on Page (124), we can be sure that two equivalent queries always build two equivalent event relations.

Lemma 18.3 (Equivalence of event relations). *If Q and \hat{Q} are two equivalent algebra queries, the two derived event relations are equivalent as well:*

$$(Q \equiv \hat{Q}) \Rightarrow (\mathbf{E}_Q \equiv \mathbf{E}_{\hat{Q}}).$$

Proof. To prove the equivalence $\mathbf{E}_Q \equiv \mathbf{E}_{\hat{Q}}$, we need to show that the extracted sets of all possible answers and the included sets of all uncovered domain classes are identical, see Definition (18.4) on Page (199).

First of all, we assume that there exists an answer tuple t that can be extracted from \mathbf{E}_Q , but not from $\mathbf{E}_{\hat{Q}}$. From Lemma (14.2) on Page (138), it follows that $t \in Q_{poss(\mathcal{W})}$ and $t \notin \hat{Q}_{poss(\mathcal{W})}$. Consequently, there is a world $W \in \mathcal{W}$, where t is in the evaluation result $Q(W)$, but not in $\hat{Q}(W)$. This contradicts $Q \equiv \hat{Q}$.

Next, we show

$$(\mathbf{D}^t \setminus \sim)^{uc} = (\hat{\mathbf{D}}^t \setminus \sim)^{uc}$$

via the following two implications:

$$\begin{aligned} \forall [d] : (([d] \in (\mathbf{D}^t \setminus \sim)^{uc}) \Rightarrow ([d] \in (\hat{\mathbf{D}}^t \setminus \sim)^{uc})) \quad \text{and} \\ \forall [d] : (([d] \in (\hat{\mathbf{D}}^t \setminus \sim)^{uc}) \Rightarrow ([d] \in (\mathbf{D}^t \setminus \sim)^{uc})). \end{aligned}$$

So, we first prove the implication

$$\forall [d] : (([d] \in (\mathbf{D}^t \setminus \sim)^{uc}) \Rightarrow ([d] \in (\hat{\mathbf{D}}^t \setminus \sim)^{uc}))$$

by contradicting its negation. Let us assume

$$\exists [d] : (([d] \in (\mathbf{D}^t \setminus \sim)^{uc}) \wedge ([d] \notin (\hat{\mathbf{D}}^t \setminus \sim)^{uc}))$$

is valid. If we fix a domain class $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$ with $[d] \notin (\hat{\mathbf{D}}^t \setminus \sim)^{uc}$, we can be sure that

$$(\hat{\mathbf{D}}^t \setminus \sim)^{uc} = (\hat{\mathbf{D}}^t \setminus \sim)^{uc} \setminus \{[d]\}.$$

In Lemma (18.1) on Page (193), we already showed that the domain value classes of $(\hat{\mathbf{D}}^t \setminus \sim)^{uc}$ are sufficient for calculating $\mathbf{P}(t \in \hat{Q})$, i.e.,

$$\mathbf{P}(t \in \hat{Q}) = \mathbf{P}(\hat{\Phi}_{\langle \bar{x} | (\hat{\mathbf{D}}^t \setminus \sim)^{uc} \rangle}^t) = \mathbf{P}(\hat{\Phi}_{\langle \bar{x} | (\hat{\mathbf{D}}^t \setminus \sim)^{uc} \setminus \{[d]\} \rangle}^t).$$

This property does not break, if we expand the set of domain values described by $(\hat{\mathbf{D}}^t \setminus \sim)^{uc} \setminus \{[d]\}$ as much as possible without taking $[d]$ into account:

$$\mathbf{P}(t \in \hat{Q}) = \mathbf{P}(\hat{\Phi}_{\langle \bar{x} | (\hat{\mathbf{D}}^t \setminus \sim)^{uc} \setminus \{[d]\} \rangle}^t) = \mathbf{P}(\hat{\Phi}_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t).$$

We simply add further (unnecessary) domain values.

In combination with $(Q \equiv \hat{Q}) \Rightarrow (\Phi^t \equiv \hat{\Phi}^t)$ (Lemma (10.1) on Page (84)), we can then conclude that

$$\mathbf{P}(t \in \hat{Q}) = \mathbf{P}(\hat{\Phi}_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) = \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) = \mathbf{P}(t \in Q).$$

However, this would mean that $\mathbf{P}(t \in Q)$ can be computed without considering the *uncovered* domain class $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$. This contradicts Property (A) of Lemma (18.1) on Page (193), which states that

$$\forall [d] \in (\mathbf{D}^t \setminus \sim) : [d] \text{ is uncovered} \Rightarrow \mathbf{P}(\Phi_{\langle \bar{x} | \mathbf{D}^t \setminus [d] \rangle}^t) \neq \mathbf{P}(t \in Q).$$

In order to prove the second implication, we simply swap the roles of the relevant domains $\hat{\mathbf{D}}^t$ and \mathbf{D}^t and apply the same arguments as used before. \square

Theorem 18.1 (Decoupled lineage optimization). *Let Q be an algebra query. If we generate two equivalent algebra queries Q_{lin} and Q_{dom} , then*

$$(Q \equiv Q_{lin} \equiv Q_{dom}) \Rightarrow (\forall t : \mathbf{P}(t \in Q) = \mathbf{P}(\mathbf{vlc}(t, Q_{lin}, \mathbf{E}_{Q_{dom}})))$$

holds.

Proof. The proposition directly follows from

$$(Q \equiv Q_{lin} \equiv Q_{dom}) \Rightarrow (\mathbf{E}_Q \equiv \mathbf{E}_{Q_{lin}} \equiv \mathbf{E}_{Q_{dom}})$$

proved in Lemma (18.3) on Page (200) and Theorem (16.1) on Page (169). \square

18.3 Exclusive worlds for uncovered domain classes

In this section, we finalize the theoretical discussions for our decoupled optimization concept by providing the last missing part of the corresponding correctness proof.

In Lemma (18.1) on Page (193), we made use of Lemma (18.4) on Page (203), which guarantees the existence of at least one *exclusive world* $W^{[d]}$ for each uncovered domain class $[d]$. To some extent, those worlds embody the witnesses for the necessity of their domain classes. Here, we present and prove:

- a set of rules, which always construct a world $W^{[d]}$ for each uncovered domain class $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$ and
- the fact that the classes $W^{[d]}$ fulfill the properties demanded by an exclusive world.

Conceptional construction rules for an exclusive world

Again, we set up construction rules over the structure of the given input query Q . Thereby, our rules often refer to the meet operation of our underlying world lattice $L(\mathcal{W})$, see Definition (18.1) on Page (188).

Definition 18.5 (Exclusive world for an uncovered domain value class). *Let Q be an algebra query and $(\mathbf{D}^t \setminus \sim)^{uc}$ be a corresponding set of all uncovered relevant domain classes. Then, we define the exclusive world for an uncovered domain class $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$ by a set of recursive rules. If we set $d = (d_{\bar{x}_1} \bullet d_{\bar{x}_2})$ with $\bar{x}_1 = \text{vars}(\text{head}(Q_1))$ and $\bar{x}_2 = \text{vars}(\text{head}(Q_2))$, we determine an exclusive world as follows:*

$$\begin{aligned}
 Q = R & : W_Q^{[d]} \text{ with } R(W_Q^{[d]}) := \{d\} \text{ and } \forall \hat{R} \in (\mathcal{R} \setminus \{R\}) : \hat{R}(W_Q^{[d]}) := \emptyset \\
 Q = \sigma_F(Q_1) & : W_Q^{[d]} := W_{Q_1}^{[d]} \\
 Q = \pi_A(Q_1) & : W_Q^{[d]} := W_{Q_1}^{[d]} \\
 Q = Q_1 \bowtie Q_2 & : W_Q^{[d]} := W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup W_{Q_2}^{[d_{\bar{x}_2}]} \\
 Q = Q_1 \cup Q_2 & : W_Q^{[d]} := \begin{cases} W_{Q_1}^{[d_{\bar{x}_1}]} & \text{if } d = (d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}) \\ W_{Q_2}^{[d_{\bar{x}_2}]} & \text{if } d = ((_)_{\bar{x}_1} \bullet d_{\bar{x}_2}) \end{cases} \\
 Q = Q_1 \setminus Q_2 & : W_Q^{[d]} := W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup \left(\bigsqcup_{\substack{[d_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \in (\mathbf{D}^t \setminus \sim)^{uc}, \\ [d_{\bar{x}_2}] \neq [d_{\bar{x}_2}]}} W_{Q_2}^{[\hat{d}_{\bar{x}_2}]} \right) \\
 Q = \rho_{(B \leftarrow A)}(Q_1) & : W_Q^{[d]} := W_{Q_1}^{[d]}.
 \end{aligned}$$

Example 18.11 (Exclusive world for an uncovered domain value class). *First, we explore our example query*

$$Q_3 = \pi_{\emptyset}(R_1 \setminus \pi_A(R_2)) \quad \text{with its subqueries } q_1 := R_1 \quad \text{and} \quad q_2 := \pi_A(R_2).$$

In particular, we intend to build the exclusive world

$$W_{Q_3}^{[d]} = W_{Q_3}^{[(1,3)]} = W_{(q_1 \setminus q_2)}^{[(1,3)]}$$

for

$$[d] = [(1,3)_{(x_A, x_B)}] = [(d_{\bar{x}_1} \bullet d_{\bar{x}_2})] = [((1)_{(x_A)} \bullet (1,3)_{(x_A, x_B)})] \in (\mathbf{D}_3^t \setminus \sim)^{uc}.$$

To set up $W_{Q_3}^{[(1,3)]} = W_{(q_1 \setminus q_2)}^{[(1,3)]}$, we basically use the difference rule of Definition (18.5) on Page (202). It says that we have to include all relevant domain classes $[d_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \in (\mathbf{D}^t \setminus \sim)^{uc}$, which differ from $[d_{\bar{x}_1} \bullet d_{\bar{x}_2}]$. Since $[\hat{d}_{\bar{x}_2}]$ is given as

$$[\hat{d}_{\bar{x}_2}] = [(1,4)_{(x_A, x_B)}] \neq [(1,3)_{(x_A, x_B)}] = [d_{\bar{x}_2}],$$

we achieve

$$W_{(q_1 \setminus q_2)}^{[(1,3)]} := W_{q_1}^{[(1)]} \sqcup W_{q_2}^{[(1,4)]} = \{(1)\} \sqcup \{(1,4)\} = \{t_1\} \sqcup \{t_4\} = W^1 \sqcup W^4 = W^7.$$

Thereby, the worlds W^1, W^4 and W^7 are taken from our lattice $L(\mathcal{W})$ of Example (18.2) on Page (188).

The determined world $W_{(q_1 \setminus q_2)}^{[(1,3)]} = W_7 = \{t_1, t_4\}$ is exclusive for our investigated domain class $[d] = [(1,3)_{(x_A, x_B)}]$, since we can list the respective substituted condition structures as follows:

$$\begin{aligned}\phi_{\mathbf{3}, \langle x_A, x_B | [(1,3)] \rangle}^{()}(W^7) &\equiv (R_1(1) \wedge \neg R_2(1, 3))(W^7) \equiv T \wedge \neg(F) \equiv T \\ \phi_{\mathbf{3}, \langle x_A, x_B | [(1,4)] \rangle}^{()}(W^7) &\equiv (R_1(1) \wedge \neg R_2(1, 4))(W^7) \equiv T \wedge \neg(T) \equiv F \\ \phi_{\mathbf{3}, \langle x_A, x_B | [(2, _)] \rangle}^{()}(W^7) &\equiv (R_1(2) \wedge \neg R_2(2, _))(W^7) \equiv F \wedge \neg(F) \equiv F.\end{aligned}$$

Therefore, $[(1,3)_{(x_A, x_B)}]$ is the only domain class of the partition $(\mathbf{D}_3^t \setminus \sim)$ that can satisfy its substituted condition structure in the world W^7 of Example (18.2) on Page (188). Accordingly, the substituted condition structure $\phi_{Q_3, \langle x_A, x_B | [(1,3)] \rangle}^{()}$ is the only disjunctive operand of

$$\Phi_{\mathbf{3}, \langle \bar{x} | \mathbf{D}_3^t \rangle}^{()} \equiv \phi_{\mathbf{3}, \langle x_A, x_B | [(1,3)] \rangle}^{()} \vee \phi_{\mathbf{3}, \langle x_A, x_B | [(1,4)] \rangle}^{()} \vee \phi_{\mathbf{3}, \langle x_A, x_B | [(2, _)] \rangle}^{()} \equiv T \vee F \vee F \equiv T$$

that can be fulfilled in the exclusive world W^7 .

This important property is exploited for proving Property (A) in the first case of Lemma (18.1) on Page (193). It implies that the domain class $[(1,3)_{(x_A, x_B)}]$ is required to select the world W^7 within the probability sum of Definition (10.10) on Page (87).

For the sake of completeness, we provide the exclusive worlds for all uncovered domain value classes of our three example queries Q_1, \dots, Q_3 :

$$\begin{aligned}Q_1 : W^{[d_1]} &= W^{[(1,1)]} = W^{[(1)]} \sqcup W^{[(1)]} = W^1 \sqcup W^1 = W^1 = \{t_1\} \\ W^{[d_3]} &= W^{[(2,2)]} = W^{[(2)]} \sqcup W^{[(2)]} = W^1 \sqcup W^2 = W^2 = \{t_2\} \\ Q_2 : W^{[d_4]} &= W^{[(1, _)]} = W^{[(_, 1)]} = W^{[(1)]} = W^1 = \{t_1\} \\ W^{[d_5]} &= W^{[(2, _)]} = W^{[(_, 2)]} = W^{[(2)]} = W^2 = \{t_2\} \\ Q_3 : W^{[d_6]} &= W^{[(1,3)]} = W^{[(1)]} \sqcup W^{[(1,4)]} = W^1 \sqcup W^4 = W^7 = \{t_1, t_4\} \\ W^{[d_7]} &= W^{[(1,4)]} = W^{[(1)]} \sqcup W^{[(1,3)]} = W^1 \sqcup W^3 = W^6 = \{t_1, t_3\} \\ W^{[d_8]} &= W^{[(2, _)]} = W^{[(2)]} = W^2 = \{t_2\}.\end{aligned}$$

Existence of exclusive worlds

Last but not least, we present our proof for the existence of at least one exclusive world for each uncovered relevant domain class.

Lemma 18.4 (Existence of exclusive worlds). *Let Q be an algebra query and*

$$Q^c \equiv \{t \mid \Phi^t\} \equiv \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t\}$$

be its equivalent domain calculus query. When $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$ is an uncovered domain class, following two properties hold for the world $W^{[d]}$ constructed by the rules of Definition (18.5) on Page (202):

Property (A): ϕ^t substituted by $[d]$ is satisfied in $W^{[d]}$:

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t(W^{[d]}) \equiv T \quad \text{and}$$

Property (B): ϕ^t substituted by $[\hat{d}]$ with $[\hat{d}] \neq [d]$ is not satisfied in $W^{[d]}$:

$$\forall [\hat{d}] \in (\mathbf{D}^t \setminus \sim)^{uc} : ([\hat{d}] \neq [d]) \Rightarrow (\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W^{[d]}) \equiv F).$$

Proof. To provide our Property (A) and (B), we apply an induction proof over the structure of Q .

I.) Induction basis (Q with $n = 1$ operators, i.e., $Q = R$):

In case of an atomic algebra query $Q = R$, our partition $(\mathbf{D}^t \setminus \sim)^{uc}$ is built by an one-to-one mapping between the tuples of $R(W^{max})$ and $(\mathbf{D}^t \setminus \sim)^{uc}$, i.e., each tuple defines its own uncovered domain class.

Then, we set the exclusive world $W^{[d]}$ to the world where the relation instance $R(W^{[d]})$ only contains the domain value d as its one and only tuple. All other relation instances are defined as empty:

$$W^{[d]} \text{ with } R(W^{[d]}) := \{d\} \quad \text{and} \quad \forall R' \in (\mathcal{R} \setminus \{R\}) : R'(W^{[d]}) := \emptyset.$$

Property (A) and (B): Obviously, only the domain value class $[d]$ can satisfy $\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv R(\bar{x})_{\langle \bar{x} | [d] \rangle}$ in the constructed world $W^{[d]} = \{d\}$, because there are no other tuples in $W^{[d]}$.

II.) Induction assumption (Q with n operators):

Let Q be an algebra query with n or less involved operations and

$$Q^c \equiv \{t \mid \Phi^t\} \equiv \{t \mid \Phi_{\langle \bar{x} | \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} | d \rangle}^t\}$$

be its equivalent domain calculus query.

When $[d] \in (\mathbf{D}^t \setminus \sim)^{uc}$ is an uncovered domain class, the following two properties hold for the world $W^{[d]}$ constructed by the rules of Definition (18.5) on Page (202):

Property (A): ϕ^t substituted by $[d]$ is satisfied in $W^{[d]}$:

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t(W^{[d]}) \equiv \text{T} \quad \text{and}$$

Property (B): ϕ^t substituted by $[\hat{d}]$ with $[\hat{d}] \neq [d]$ is not satisfied in $W^{[d]}$:

$$\forall [\hat{d}] \in (\mathbf{D}^t \setminus \sim)^{uc} : ([\hat{d}] \neq [d]) \Rightarrow (\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W^{[d]}) \equiv \text{F}).$$

III.) Induction step (Q with $n + 1$ operators):

In the following, we verify six cases inferred from the six remaining algebra operators. As before, we set the variable sets \bar{x}, \bar{x}_1 , and \bar{x}_2 to $\text{vars}(\phi_Q^t)$, $\text{vars}(\phi_{Q_1}^t)$, and $\text{vars}(\phi_{Q_2}^t)$.

Case 1: $Q = \sigma_F(Q_1)$

Using the MAC rules of Lemma (10.1) on Page (84), we achieve the substituted condition structure:

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv (\phi_{Q_1}^t \wedge F)_{\langle \bar{x} | [d] \rangle}.$$

The induction assumption assures that there is an exclusive world $W_{Q_1}^{[d]}$ for the relevant domain class $[d]$, since $[d] \in (\mathbf{D}_{Q_1}^t \setminus \sim)^{uc}$. We use this exclusive world directly in the following manner:

$$W_Q^{[d]} := W_{Q_1}^{[d]}.$$

Property (A): Since the domain class $[d]$ is part of $(\mathbf{D}_Q^t \setminus \sim)^{uc}$, we know that $[d]$ is relevant and the selection condition must be fulfilled, i.e., $F_{\langle \bar{x} | [d] \rangle} \equiv \text{T}$. The capability of satisfying

the selection condition $F_{\langle \bar{x} \mid [d] \rangle}$ is independent from a specific world. Then, it follows from the induction assumption and our relation rule of Definition (18.5) on Page (202) that:

$$\begin{aligned} (\phi_{Q_1}^t \wedge F)_{\langle \bar{x} \mid [d] \rangle}(W_Q^{[d]}) &\equiv (\phi_{Q_1, \langle \bar{x} \mid [d] \rangle}^t(W_Q^{[d]}) \wedge F_{\langle \bar{x} \mid [d] \rangle}) \\ &\equiv (\phi_{Q_1, \langle \bar{x} \mid [d] \rangle}^t(W_Q^{[d]}) \wedge \top) \\ &\equiv (\phi_{Q_1, \langle \bar{x} \mid [d] \rangle}^t(W_{Q_1}^{[d]}) \wedge \top) \\ &\equiv \top \wedge \top \equiv \top. \end{aligned}$$

Property (B): From the induction assumption, we know that the world $W_Q^{[d]} = W_{Q_1}^{[d]}$ is exclusive for $[d]$. Consequently, all remaining uncovered domain classes $[\hat{d}]$ fail on $\phi_{Q_1, \langle \bar{x} \mid [\hat{d}] \rangle}^t$, see Property (B) of our induction assumption:

$$\begin{aligned} (\phi_{Q_1}^t \wedge F)_{\langle \bar{x} \mid [\hat{d}] \rangle}(W_Q^{[d]}) &\equiv (\phi_{Q_1, \langle \bar{x} \mid [\hat{d}] \rangle}^t(W_Q^{[d]}) \wedge F_{\langle \bar{x} \mid [\hat{d}] \rangle}) \\ &\equiv (\phi_{Q_1, \langle \bar{x} \mid [\hat{d}] \rangle}^t(W_Q^{[d]}) \wedge \top) \\ &\equiv (\phi_{Q_1, \langle \bar{x} \mid [\hat{d}] \rangle}^t(W_{Q_1}^{[d]}) \wedge \top) \\ &\equiv \text{F} \wedge \top \equiv \text{F}. \end{aligned}$$

Case 2: $Q = \pi_A(Q_1)$

On the basis of the MAC rules of Lemma (10.1) on Page (84), we consider the substituted condition structure

$$\phi_{Q, \langle \bar{x} \mid [d] \rangle}^t \equiv \phi_{Q_1, \langle \bar{x} \mid [d] \rangle}^t.$$

We can directly use the exclusive world provided by the induction assumption, i.e.,

$$W_Q^{[d]} := W_{Q_1}^{[d]}.$$

Property (A) and (B): Our claimed properties directly follow from the induction assumption.

Case 3: $Q = Q_1 \bowtie Q_2$

We rely on the MAC rules of Lemma (10.1) on Page (84), which allows us to explore the substituted condition structure

$$\phi_{Q, \langle \bar{x} \mid [d] \rangle}^t \equiv (\phi_{Q_1}^t \wedge \phi_{Q_2}^t)_{\langle \bar{x} \mid [d] \rangle} \equiv (\phi_{Q_1, \langle \bar{x}_1 \mid [d_{\bar{x}_1}] \rangle}^t) \wedge (\phi_{Q_2, \langle \bar{x}_2 \mid [d_{\bar{x}_2}] \rangle}^t).$$

Furthermore, we know that $[d_{\bar{x}_1}]$ and $[d_{\bar{x}_2}]$ are relevant, since $[d] = [d_{\bar{x}_1} \bullet d_{\bar{x}_2}]$ belongs to $(\mathbf{D}_Q^t \setminus \sim)^{uc}$.

By taking advantage of our induction assumption, we employ the two exclusive worlds $W_{Q_1}^{[d_{\bar{x}_1}]}$ and $W_{Q_2}^{[d_{\bar{x}_2}]}$ in order to set up $W_Q^{[d]}$. More concretely, we unify all given relation instances in the form of

$$W_Q^{[d]} := W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup W_{Q_2}^{[d_{\bar{x}_2}]} \quad \text{with} \quad Q_1 \not\equiv Q_2.$$

Otherwise, we simplify Q to

$$Q \equiv Q_1 \bowtie Q_2 \equiv Q_1 \bowtie Q_1 \equiv Q_1$$

and apply our induction assumption directly.

Property (A): The substituted condition structure

$$(\phi_{Q_1}^t \wedge \phi_{Q_2}^t)_{\langle \bar{x} \mid [d] \rangle}(W_Q^{[d]}) \equiv (\phi_{Q_1}^t \wedge \phi_{Q_2}^t)_{\langle \bar{x} \mid [d] \rangle}(W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup W_{Q_2}^{[d_{\bar{x}_2}]})$$

$$\begin{aligned}
&\equiv \phi_{Q_1, \langle \bar{x}_1 | d_{\bar{x}_1} \rangle}^t(W_{Q_1}^{[d_{\bar{x}_1}]}) \wedge \phi_{Q_2, \langle \bar{x}_2 | d_{\bar{x}_2} \rangle}^t(W_{Q_2}^{[d_{\bar{x}_2}]}) \\
&\equiv \top \wedge \top
\end{aligned}$$

is obviously fulfilled, since our constructed world $W_Q^{[d]} = W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup W_{Q_2}^{[d_{\bar{x}_2}]}$ contains all tuples from $W_{Q_1}^{[d_{\bar{x}_1}]}$ and $W_{Q_2}^{[d_{\bar{x}_2}]}$ that are needed to satisfy $\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t$ and $\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t$ simultaneously.

Property (B): Here, we need to show that $[d]$ is the only uncovered domain class which can be fulfilled in $W^{[d]}$. Therefore, we investigate the negation of Property (B). So, let us assume that

$$\exists [\hat{d}] \in (\mathbf{D}_Q \setminus \sim)^{uc} : ([\hat{d}] \neq [d]) \wedge (\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W_Q^{[\hat{d}]})$$

is given. This leads to the following contradiction:

$$\begin{aligned}
\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W_Q^{[\hat{d}]}) &\Rightarrow (\phi_{Q_1, \langle \bar{x}_1 | [\hat{d}_{\bar{x}_1}] \rangle}^t \wedge \phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t)(W_{Q_1}^{[\hat{d}_{\bar{x}_1}]} \sqcup W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}) \\
&\Rightarrow (\phi_{Q_1, \langle \bar{x}_1 | [\hat{d}_{\bar{x}_1}] \rangle}^t(W_{Q_1}^{[\hat{d}_{\bar{x}_1}]}) \wedge (\phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t(W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}) \quad (Q_1 \neq Q_2) \\
&\Rightarrow ([\hat{d}_{\bar{x}_1}] = [d_{\bar{x}_1}]) \wedge ([\hat{d}_{\bar{x}_2}] = [d_{\bar{x}_2}]) \quad (\text{IA}) \\
&\Rightarrow [d] = [\hat{d}] \quad \text{!}
\end{aligned}$$

Case 4: $Q = Q_1 \cup Q_2$

In accordance with the union rule for constructing relevant domains, (Definition (12.1)) on Page (109), we make a distinction between these two cases:

- $[d] = [d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}]$ with $[d] \subseteq \mathcal{M}_1$ and
- $[d] = [(_)_{\bar{x}_1} \bullet d_{\bar{x}_2}]$ with $[d] \subseteq \mathcal{M}_2$.

Case 4.1: $Q = Q_1 \cup Q_2$ with domain values of the form $[d] = [d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}]$

By employing the MAC rules of Lemma (10.1) on Page (84) in conjunction with Lemma (14.3) on Page (138), we simplify $\phi_{Q, \langle \bar{x} | [d] \rangle}^t$ to $\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t$:

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv \phi_{Q, \langle \bar{x} | [d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}] \rangle}^t \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \vee \phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t) \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \vee \text{F}) \equiv \phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t.$$

Thus, $[d]$ is relevant, if $[d_{\bar{x}_1}]$ is relevant. Accordingly, we can use the exclusive world $W_{Q_1}^{[d_{\bar{x}_1}]}$ provided by the induction assumption in order to build $W_Q^{[d]}$, i.e.,

$$W_Q^{[d]} := W_{Q_1}^{[d_{\bar{x}_1}]}.$$

Property (A) and (B): Both properties directly follow from the induction assumption, since

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t(W_Q^{[d]}) \equiv \phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t(W_{Q_1}^{[d_{\bar{x}_1}]}) \equiv \phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t(W_{Q_1}^{[d_{\bar{x}_1}]}).$$

Case 4.2: $Q = Q_1 \cup Q_2$ with domain values of the form $[d] = [(_)_{\bar{x}_1} \bullet d_{\bar{x}_2}]$

The proof of this case is analog to Case (4.1). We just swap the roles of $\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t$ and $\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t$.

Case 5: $Q = Q_1 \setminus Q_2$

A relevant domain value produced by the difference rule of Definition (13.3) on Page (125) can have two different forms: $[d] = [d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}]$ and $[d] = [d_{\bar{x}_1} \bullet d_{\bar{x}_2}]$. We treat both instances as two separate cases.

Case 5.1: $Q = Q_1 \setminus Q_2$ with domain value of the form $[d] = [d_{\bar{x}_1} \bullet (_)_{\bar{x}_2}]$

The MAC rules of Lemma (10.1) on Page (84) in conjunction with Lemma (14.3) on Page (138) allow us to transform $\phi_{Q, \langle \bar{x} | [d] \rangle}^t$ to

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | (_)_{\bar{x}_2} \rangle}^t)) \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \wedge \neg(F)) \equiv \phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t.$$

Since $[d_{\bar{x}_1}]$ is relevant for $\Phi_{Q_1}^t$, we can employ the exclusive world $W_{Q_1}^{[d_{\bar{x}_1}]}$ provided by the induction assumption to build $W_Q^{[d]}$, i.e., $W_Q^{[d]} := W_{Q_1}^{[d]}$.

Please note that $W_{Q_1}^{[d]}$ equals the constructed world of Definition (18.5) on Page (202), i.e.,

$$W_{Q_1}^{[d]} = W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup \left(\bigsqcup_{\substack{[d_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \in (\mathbf{D}^t \setminus \sim)^{uc}, \\ [\hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_2}]}} W_{Q_2}^{[\hat{d}_{\bar{x}_2}]} \right),$$

because by construction there is no $[\hat{d}_{\bar{x}_2}]$ with $[\hat{d}_{\bar{x}_2}] \neq (_)_{\bar{x}_2}$ in this case.

Property (A) and (B): The desired properties directly follow from the induction assumption, since

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv \phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t.$$

Case 5.2: $Q = Q_1 \setminus Q_2$ with domain value of the form $[d] = [d_{\bar{x}_1} \bullet d_{\bar{x}_2}]$

The MAC rules of Lemma (10.1) on Page (84) lead to the condition structure

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t)).$$

We construct our exclusive $W^{[d]}$ on the basis of several exclusive worlds that are all provided by our induction assumption. Besides $W_{Q_1}^{[d_{\bar{x}_1}]}$, we additionally insert the tuples of all worlds $W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}$ with $[d_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \in (\mathbf{D}^t \setminus \sim)^{uc}$ and $[\hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_2}]$ into $W_Q^{[d]}$:

$$W_Q^{[d]} := W_Q^{[d_{\bar{x}_1} \bullet d_{\bar{x}_2}]} = W_{Q_1}^{[d_{\bar{x}_1}]} \sqcup \left(\bigsqcup_{\substack{[d_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \in (\mathbf{D}^t \setminus \sim)^{uc}, \\ [\hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_2}]}} W_{Q_2}^{[\hat{d}_{\bar{x}_2}]} \right).$$

Property (A): The analysis of

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t(W^{[d]}) \equiv (\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t))(W_Q^{[d]})$$

shows that the first operand $\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t$ of $\phi_{Q, \langle \bar{x} | [d] \rangle}^t$ is obviously fulfilled, since we include all that are needed to satisfy $\phi_{Q_1, \langle \bar{x}_1 | [d_{\bar{x}_1}] \rangle}^t$ from $W_{Q_1}^{[d_{\bar{x}_1}]}$ into $W_Q^{[d]}$.

Furthermore, $\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t$ cannot be satisfied by the tuples from $W_{Q_1}^{[d_{\bar{x}_1}]}$, since we can assume $Q_1 \neq Q_2$. In the case of $Q_1 \equiv Q_2$, we would achieve the following entities:

$$Q \equiv Q_1 \setminus Q_2 = \emptyset \quad \text{with} \quad \mathbf{D}^t = \emptyset \quad \text{and} \quad (\mathbf{D}^t \setminus \sim)^{uc} = \emptyset.$$

The second operand $\phi_{Q_2, \langle \bar{x}_2 | [d_{\bar{x}_2}] \rangle}^t$ cannot be fulfilled in a world $W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}$ with $[d_{\bar{x}_2}] \neq [\hat{d}_{\bar{x}_2}]$ either, because $W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}$ is exclusive for $[\hat{d}_{\bar{x}_2}]$.

Property (B): Here, we have to show that

$$\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W^{[d]}) \equiv (\phi_{Q_1, \langle \bar{x}_1 | [\hat{d}_{\bar{x}_1}] \rangle}^t \wedge \neg(\phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t))(W^{[d]}) \equiv \text{F},$$

if

$$[\hat{d}] = [\hat{d}_{\bar{x}_1} \bullet \hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_1} \bullet d_{\bar{x}_2}] = [d].$$

Clearly, $[\hat{d}]$ differs from $[d]$, when $[\hat{d}_{\bar{x}_1}] \neq [d_{\bar{x}_1}]$ and/or $[\hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_2}]$. In case of $[\hat{d}_{\bar{x}_1}] \neq [d_{\bar{x}_1}]$, we can immediately conclude that $\phi_{Q, \langle \bar{x} | [\hat{d}] \rangle}^t(W^{[d]})$ fails, since the first conjunctively combined operand $\phi_{Q_1, \langle \bar{x}_1 | [\hat{d}_{\bar{x}_1}] \rangle}^t$ already fails in $W_{Q_1}^{[d_{\bar{x}_1}]}$, see Property (B) of induction assumption.

In contrast, when $[\hat{d}_{\bar{x}_1}] = [d_{\bar{x}_1}]$, the first operand is fulfilled and $[\hat{d}_{\bar{x}_2}] \neq [d_{\bar{x}_2}]$. Thus, we need to verify that

$$(\neg(\phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t))(W^{[d]}) \equiv \text{F} \Leftrightarrow (\phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t)(W^{[d]}) \equiv \text{T}.$$

The subcondition $(\phi_{Q_2, \langle \bar{x}_2 | [\hat{d}_{\bar{x}_2}] \rangle}^t)(W^{[d]}) \equiv \text{T}$ holds, because our construction rule inserts all required tuples provided by the worlds $W_{Q_2}^{[\hat{d}_{\bar{x}_2}]}$ into $W^{[d]}$.

Case 6: $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$

In this case, we deal with the substituted condition structure

$$\phi_{Q, \langle \bar{x} | [d] \rangle}^t \equiv (\phi_{Q_1, \langle \mathcal{B} | \mathcal{A} \rangle}^t)_{\langle \bar{x} | [d] \rangle}$$

in accordance with the MAC rule for a renaming operation, see Lemma (10.1) on Page (84).

Constructing $W^{[d]}$: We directly use the exclusive world provided by our induction assumption, i.e., $W_Q^{[d]} := W_{Q_1}^{[d]}$.

Property (A) and (B): The properties directly follow from our induction assumption. □

Example 18.12 (Exclusive worlds). *In Example (18.11) on Page (202), we created the exclusive worlds for our example queries Q_1, Q_2 , and Q_3 of Example (18.1) on Page (187).*

Not surprisingly, we end up with the same worlds, if we apply our construction rules of Definition (18.5) on Page (202) on the uncovered domain classes of their equivalent counterparts $Q_{1, \text{dom}}, Q_{2, \text{dom}}$, and $Q_{3, \text{dom}}$ of Example (18.9) on Page (197):

$$\begin{aligned} Q_{1, \text{dom}} : W^{[(1,1)]} &= W^{[(1)]} \sqcup W^{[(1)]} = W^1 \sqcup W^1 = W^1 = W^{[d_1]} \\ W^{[(2,2)]} &= W^{[(2)]} \sqcup W^{[(2)]} = W^1 \sqcup W^2 = W^2 = W^{[d_3]} \\ Q_{2, \text{dom}} : W^{[(1)]} &= W^1 = W^{[d_4]} \\ W^{[(2)]} &= W^2 = W^{[d_5]} \\ Q_{3, \text{dom}} : W^{[(1,3, _)]} &= W^{[(1)]} \sqcup W^{[(4)]} = W^1 \sqcup W^4 = W^7 = W^{[d_6]} \\ W^{[(1,4, _)]} &= W^{[(1)]} \sqcup W^{[(3)]} = W^1 \sqcup W^3 = W^6 = W^{[d_7]} \\ W^{[(2, _, _)]} &= W^{[(2)]} = W^2 = W^{[d_8]} \\ W^{[(1,3,1)]} &= (W^{[(1)]} \sqcup W^{[(4)]}) \sqcup W^{[(1)]} = (W^1 \sqcup W^4) \sqcup W^1 = W^7 \sqcup W^1 = W^7 = W^{[d_6]} \\ W^{[(1,4,1)]} &= (W^{[(1)]} \sqcup W^{[(3)]}) \sqcup W^{[(1)]} = (W^1 \sqcup W^3) \sqcup W^1 = W^6 \sqcup W^1 = W^6 = W^{[d_7]} \\ W^{[(2, _, 2)]} &= W^{[(2)]} \sqcup W^{[(2)]} = W^2 \sqcup W^2 = W^2 = W^{[d_8]}. \end{aligned}$$

Chapter 19

Lineage Factorization and Compression

In this chapter, we present the third key technique of our framework, namely a novel method for lineage factorization and compression. Although we can seamlessly integrate all our main contributions into one single algorithm, we first describe our factorization and compression idea as a distinct concept. This chapter consists of the following sections:

- Section (19.1): existing state-of-the-art approaches for lineage factorization and compression,
- Section (19.2): our λ -labeling factorization and compression method, which still uses the classical lineage construction rules of Lemma (6.1) on Page (35),
- Section (19.3): the query class for which our technique can identify *all* shared lineage formulas,
- Section (19.4): the integration of our new technique into the vertical construction algorithm, and
- Section (19.5): our final series of experiments.

19.1 Existing lineage factorization and compression methods

Beyond the actual construction process, we are also interested in the enhancement of our central data structures representing lineage formulas. To motivate the compression of lineage formulas, we revisit the lineage formulas constructed in Example (8.3) on Page (56).

Example 19.1 (Shared lineage subformulas). *Let us again have look at our lineage formula*

$$\varphi_{Q_L}^{USA} = ((\underline{e_4 \wedge e_9}) \wedge e_{11}) \oslash (((e_5 \oslash e_6) \wedge e_{10}) \wedge e_{13})$$

and

$$\varphi_{Q_L}^{Italy} = (\underline{e_4 \wedge e_9}) \wedge e_{12}$$

of Example (8.3) on Page (56). We easily recognize that there are subformulas that are shared between $\varphi_{Q_L}^{USA}$ and $\varphi_{Q_L}^{Italy}$, e.g., $\underline{e_4}$ and $\underline{(e_4 \wedge e_9)}$.

By following the conventions introduced in [101] and [84], we denote each lineage *subformula* as a *factor* of the overall lineage formula. Please note that we use the term *factor* to describe both a lineage formula and a network random variable (Section (8.4) and Figure (8.4) on Page (57)) interchangeably. In both formalisms, a lineage formula is recursively represented by a set of factors, which stand for its lineage subformulas.

Furthermore, a specific factor is said to be a *shared factor*, if it is occurring more than once in a set of lineage formulas. Previous works already demonstrated that shared factors can be effectively exploited to build very compact data structures for lineage formulas, e.g., compressed factor networks and formula trees [101, 84].

A crucial point of lineage compression is the identification of as many *shared* factors as possible. This process is also known as *lineage factorization*.

Analogously to lineage optimization techniques described earlier, we can approach lineage factorization on query and/or lineage level:

- Lineage factorization directly on lineage formulas: An obvious method for lineage factorization is to compare the structures of all given lineage subformulas. A mechanism following and improving this basic idea is used in the PrDB system [101]. It directly factorizes already constructed lineage formulas.
- Lineage factorization indirectly by special query plans: Olteanu and Wen showed in [84] that shared factors can also be determined indirectly by only analyzing the given input query. Their probabilistic query SPROUT2 rewrites the given input query into a *share plan* in order to identify factors [84].

Next, we briefly describe both methods in more detail.

Lineage factorization performed directly on lineage formulas

In Figure (8.4) on Page (57), we already illustrated that lineage formulas can be equivalently expressed by a factor network. Using those networks, Sen, Deshpande and Getoor developed in [101] a compression technique as a key feature of their shared inference framework. More specifically, they applied a customized bisimulation algorithm for minimizing the underlying graph structures of their networks.

We discussed a typical factor network used in PrDB in Section (8.4). It represented our example lineage formulas $\varphi_{Q_L}^{USA}$ and $\varphi_{Q_L}^{Italy}$. In general, a specific factor is denoted as $f_{id}^{type}(op_1, \dots, op_n)$, where

- id is a unique number assigned during the construction process,
- $type$ gives the top-most logical operator of the corresponding lineage subformula and
- op_1, \dots, op_n gives the factor operands.

In order to compare a specific pair of factors, PrDB does not check their syntactic structures directly. Instead, it computes special *labels*¹ for comparing factors. The label of a given factor is derived from its type and the types of its operands by a recursive label function λ :

$$\lambda(f_{id}^{type}(op_1, \dots, op_n)) := \begin{cases} \{e_i\} & \text{if } id = e_i \\ \{type, \lambda(op_1), \dots, \lambda(op_n)\} & \text{else.} \end{cases}$$

Example 19.2 (Comparison of factors based on labels). *Let us artificially separate all factor identifiers used in the factor representations of $\varphi_{Q_L}^{USA}$ and $\varphi_{Q_L}^{Italy}$ of Example (8.4) on Page (57):*

$$\varphi_{Q_L}^{USA} \equiv f_{13}^{\odot}(f_{10}^{\wedge}(\underline{f_8^{\wedge}(f_2^{\odot}(f_{e_4}), f_5^{\odot}(f_{e_9}))}, f_{e_{11}}), f_{12}^{\wedge}(f_9^{\wedge}(f_3^{\odot}(f_{e_5}, f_{e_6}), f_6^{\odot}(f_{e_{10}})), f_{e_{13}}))$$

and

$$\varphi_{Q_L}^{Italy} \equiv f_{14'}^{\odot}(f_{11'}^{\wedge}(f_{8'}^{\wedge}(f_{2'}^{\odot}(f_{e_4}), f_{5'}^{\odot}(f_{e_9})), f_{e_{12}})).$$

¹Please note that PrDB originally uses the term *key* instead of *label*. For the sake of comparison, we adapt the used terminology.

Such a configuration can be easily found, when identical subqueries occur more than once in a given input query. Please notice that the underlined network part represents again our shared factor $(e_4 \wedge e_9)$ of Example (19.2) on Page (210).

In spite of having different factor identifiers in place, PrDB is still able to recognize that the lineage subformulas

$$f_8^\wedge(f_2^\odot(f_{e_4}), f_5^\odot(f_{e_9})) \quad \text{and} \quad f_{8'}^\wedge(f_{2'}^\odot(f_{e_4}), f_{5'}^\odot(f_{e_9}))$$

embody the same shared factor. To do so, it only needs to match their identical factor labels:

$$\lambda(f_8^\wedge(f_2^\odot(f_{e_4}), f_5^\odot(f_{e_9}))) = \{\wedge, \{\odot, \{e_4\}\}, \{\odot, \{e_9\}\}\} = \lambda(f_{8'}^\wedge(f_{2'}^\odot(f_{e_4}), f_{5'}^\odot(f_{e_9}))).$$

In a nutshell, the PrDB algorithm checks all pairs of factors which have the same number of (recursive) operands. Obviously, PrDB's factor labels indirectly capture the underlying syntactic lineage structures. Therefore, all shared factors can be found by this method. Moreover, it is applicable on lineage formulas built for all kinds of **SPJUD**-queries on arbitrary probabilistic databases.

Besides these positive benefits, the PrDB factorization also suffers from two notable downsides:

- First and foremost, the generation of all factor labels is relatively expensive. In fact, we achieve a quadratic complexity for computing all labels, if we take into account that the label length based on an n -ary logical operator is already linear.
- Secondly, the proposed algorithm of [101] cannot compare factor labels during the construction of a factor network, since it needs to generate all operand keys recursively. In other words, it always sets up the entire network before it starts to compress it again.

The measured execution times of our experiments clearly showed the negative impacts of these drawbacks, see Chapter (19.5).

Lineage factorization based on query rewriting

A further approach for lineage factorization has been published by Olteanu and Wen in [84]. In contrast to all other techniques discussed so far, they did not investigate the problem of query evaluation of Definition (5.2) on Page (30). Instead, they were interested in the *ranking* of answer tuples based on their computed probabilities. To tackle this task, Olteanu and Wen rewrote the input query into a so-called *share plan*.

The main goal of share plans is to identify shared factors produced by a hard *subquery* of the given input query. Olteanu and Wen proved for a certain class of queries that those intractable factors can be neglected for determining a ranking of all answer tuples. Additionally, they presented a remarkable theoretical result in the form of a dichotomy between tractable and $\#\mathcal{P}$ -hard **nrSPJ**-queries applied on TID databases.

Notably, share plans, which has to be performed within an RDBMS [84], also enforce strict orderings of projection and join operations. In Section (17.1), we already pointed out the disadvantages of such fixed operation patterns and sequences in the context of safe plans. For instance, our example query Q_L of Example (17.1) on Page (173) represents a safe plan as well as a share plan.

Beyond inefficient operation orderings, share plans are also impaired by demanding a large set of overlapping join attributes in order to cause a serious effect [84]. Unfortunately, queries that involve union and difference operations are not supported at all.

Design goals

Analogously to our former key techniques, we state a set of design goals, which are inferred from the positive and negative properties of the related existing methods:

Fulfillment of design goals for lineage factorization and compression			
	design goal	share plans (SPROUT2)	bisimulation (PrDB)
1	full relational algebra support	no	yes
2	finding all shared factors (query class)	tractable SPJ	SPJUD
3	efficient label computation	-	no
4	on-the-fly factorization	no	no

Figure 19.1: Fulfillment of design goals for lineage factorization and compression

- Design goal (1): full relational algebra support,
- Design goal (2): finding all shared factors (query class),
- Design goal (3): efficient label computation, and
- Design goal (4): on-the-fly factorization.

Figure (19.1) on Page (212) depicts how our studied approaches fulfill the stated design goals.

19.2 Lineage factorization by λ -labeling

In this section, we outline the main idea behind our lineage factorization and compression method.

Its main idea consists of three consecutive steps:

- *construction phase*: building of all lineage formulas in form of formula trees,
- *labeling phase*: annotation of all formula subtrees with special labels and
- *merging phase*: merging of identical labeled subtrees.

The outcome of all three processing steps is a set of compressed formula trees, which embody the final lineage formulas. Before describing these phases in more depth, we exemplify the benefits that can be achieved by means of an additional factorization and compression step.

Example 19.3 (Example query for lineage factorization). *Throughout this chapter, we explore a new example query Q_1 :*

$$Q_1 = R_2 \setminus (R_2 \bowtie R_4).$$

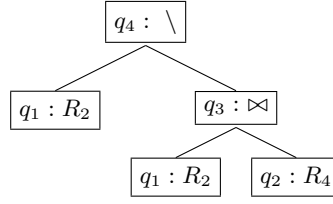
It is evaluated on our running example database of Example (4.1) on Page (24). The query tree of Q_1 is additionally depicted in Figure (19.2) on Page (213).

At this point, we still make use of the classical construction rules of Lemma (6.1) on Page (35) in order to determine the lineage formulas for Q_1 . By doing so, we obtain following two lineage formulas:

$$\begin{aligned}\varphi_1^{(1,3)} &= e_3 \wedge \neg(e_3 \wedge e_9) \\ \varphi_1^{(1,4)} &= e_4 \wedge \neg(e_4 \wedge e_9).\end{aligned}$$

They are constructed for the two answer tuples $(1, 3)_{(A,B)}$ and $(1, 4)_{(A,B)}$ of Q_1 .

A closer look at the structures of $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ reveals that subformulas of $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ occur multiple times, e.g., e_3 and e_4 appear twice in $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$. The subformula e_9 is even shared between both lineage formulas.

Figure 19.2: Example query Q_1 with its subquery identifier q_1, \dots, q_4

Analogously to our **vlc**-algorithm, we assume that the classical construction rules build lineage formulas in form of formula trees. Figure (19.3) on Page (214) depicts the two formula trees built for $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$. If we count all nodes used in both formula trees, we obtain a total number of $2 * 5 = 10$ tree nodes.

Next, let us imagine we have a data structure, where each unique subformula is represented only once. This means that subformulas as e_3 and e_4 would be materialized by only one subtree instance. In this case, we can reduce the overall number of nodes from 10 to 7. Such a compression can lead to a considerable memory saving and an increased processing speed up, since all tasks performed on repeating subtrees before can be accomplished in one traversing step.

Our experiments in Section (19.5) clearly underpin these improvements for constructing and evaluating compressed formula trees.

Next, we give a precise specification of the essential term *factor*.

Definition 19.1 (Factor of a lineage formula). *Let $\{\varphi^t\}$ be a set of lineage formulas constructed for a given algebra query Q .*

- Then, each lineage subformula of a lineage formula out of $\{\varphi^t\}$ is called a factor.
- If a specific factor occurs more than once within a single lineage formula or between several lineage formulas, it is said to be a shared factor.
- The set of all factors derived from a lineage formula φ^t is denoted as $\mathcal{F}(\varphi^t)$.

Example 19.4 (Factors of lineage formulas). *If we take into account the lineage formulas*

$$\varphi_1^{(1,3)} = e_3 \wedge \neg(e_3 \wedge e_9) \quad \text{and} \quad \varphi_1^{(1,4)} = e_4 \wedge \neg(e_4 \wedge e_9)$$

of Example (19.3) on Page (212), the corresponding sets of all factors are given as

$$\begin{aligned} \mathcal{F}(\varphi_1^{(1,3)}) &= \{e_3, e_9, (e_3 \wedge e_9), e_3 \wedge \neg(e_3 \wedge e_9)\} \\ \mathcal{F}(\varphi_1^{(1,4)}) &= \{e_4, e_9, (e_4 \wedge e_9), e_4 \wedge \neg(e_4 \wedge e_9)\}. \end{aligned}$$

The factors e_3, e_4 , and e_9 are shared factors, since they occur multiple times within/between $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$.

Constructing phase

First of all, we need to construct all lineage formulas, before we can factorize and compress them. Please be aware that we are still describing our factorization and compression method *without* using our **vlc**-algorithm. Thus, we simply utilize the classical rules of Lemma (6.1) on Page (35).

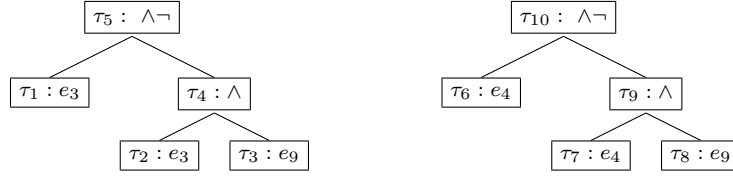


Figure 19.3: Classical formula trees for $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ with subtree identifiers

In Chapter (15), we already worked with formula trees representing lineage formulas.

In order to address specific subtrees of an overall formula tree, we assign a unique subtree identifier τ to each subtree root node.

Definition 19.2 (Formula tree of lineage formula). *Let φ^t be a lineage formula represented by a formula tree.*

- Then, we identify each subtree of φ^t with a unique subtree identifier τ .
- The overall formula tree for φ^t is then described by its set of all subtrees:

$$\mathcal{T}(\varphi^t) = \{\tau_1, \dots, \tau_k\}.$$

- The represented factor of a specific subtree τ is denoted as

$$\varphi(\tau).$$

Example 19.5 (Formula tree of lineage formula). *By applying Definition (19.2) on Page (214), we can describe the formula trees of $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ as*

$$\mathcal{T}(\varphi_1^{(1,3)}) = \{\tau_1, \dots, \tau_5\} \quad \text{and} \quad \mathcal{T}(\varphi_1^{(1,4)}) = \{\tau_6, \dots, \tau_{10}\},$$

see Figure (19.3) on Page (214). Moreover, we can refer to the factors which are encoded by the subtrees τ_1 , τ_3 , and τ_9 as follows:

$$\varphi(\tau_1) = e_3, \quad \varphi(\tau_3) = e_9 \quad \text{and} \quad \varphi(\tau_9) = e_4 \wedge e_9.$$

Please note that different subtrees can represent the same factor, e.g.,

$$\varphi(\tau_1) = e_3 = \varphi(\tau_2) \quad \text{and} \quad \varphi(\tau_3) = e_9 = \varphi(\tau_8).$$

Labeling phase

Next, we describe the second phase of our lineage factorization and compression mechanism. So far, we have constructed formula trees, which embody our lineage formulas. Thereby, a built formula tree consists of a set of formula subtrees that represents the factors of Definition (19.4) on Page (213).

Subsequently, we introduce a specific label for each subtree.

Our subtree labeling guarantees that two factors annotated by the same label are always syntactically identical.

We employ this important property in Section (19.2) in order to remove redundant subtrees from our final formula trees.

In Section (19.1), we showed that PrDB creates the label of a specific factor by recursively using the already computed labels of its operands. So, the computation of a single label can already involve as many operands as the given input tuples. Our labeling idea is different.

On the basis of information describing the underlying query evaluation process, we generate a subtree label.

An essential part of a subtree label is the subquery identifier, which points to the specific intermediate query result created by the corresponding *subquery* of Q .

Definition 19.3 (Set of all subqueries $\mathcal{SQ}(Q)$). *Let Q be an algebra query. Then, we collect all syntactically different subqueries of Q in the set of all subqueries:*

$$\mathcal{SQ}(Q) = \{q_1, \dots, q_n\}.$$

Example 19.6 (Set of all subqueries $\mathcal{SQ}(Q)$). *For our query Q_1 of Example (19.3) on Page (212), we obtain the following set of subquery identifiers:*

$$\mathcal{SQ}(Q_1) = \{q_1, \dots, q_4\}$$

with

$$\begin{aligned} q_1 &:= R_2 & q_3 &:= R_2 \bowtie R_4 \\ q_2 &:= R_4 & q_4 &:= R_2 \setminus (R_2 \bowtie R_4) \end{aligned}$$

as illustrated in Figure (19.2) on Page (213).

Remark 19.1 (Identical vs. equivalent subqueries). *We emphasize that our set of all subqueries $\mathcal{SQ}(Q)$ can include different identifiers for equivalent subqueries. As an alternative to Definition (19.3) on Page (215), we could try to define a unique subquery identifier for each class of equivalent subqueries. We leave the problem of finding all equivalent subqueries of an arbitrary query from SPJUD out of our considerations.*

Using our newly introduced subquery identifiers, we next create a label λ_q^t for each subtree τ of $\mathcal{T}(\varphi^t)$. The notation λ_q^t indicates that the labeled subtree encodes a factor, which once belonged to an answer tuple t contained in the intermediate query result of q .

We formalize our idea of subtree labeling by means of an injective function between the set of subtrees $\mathcal{T}(\varphi^t)$ and the general set of all subtree labels Λ .

Intermediate result of $q_1 = R_2$			
A	B	factor	label
1	3	$\varphi(\tau_1) = \varphi(\tau_2) = e_3$	$\lambda(\tau_1) = \lambda(\tau_2) = \lambda_{q_1}^{(1,3)}$
1	4	$\varphi(\tau_6) = \varphi(\tau_7) = e_4$	$\lambda(\tau_6) = \lambda(\tau_7) = \lambda_{q_1}^{(1,4)}$

Intermediate result of $q_2 = R_4$		
A	factor	label
1	$\varphi(\tau_3) = \varphi(\tau_8) = e_9$	$\lambda(\tau_3) = \lambda(\tau_8) = \lambda_{q_2}^{(1)}$
2	-	-
3	-	-

Intermediate result of $q_3 = R_2 \bowtie R_4$			
A	B	factor	label
1	3	$\varphi(\tau_4) = e_3 \wedge e_9$	$\lambda(\tau_4) = \lambda_{q_3}^{(1,3)}$
1	4	$\varphi(\tau_9) = e_4 \wedge e_9$	$\lambda(\tau_9) = \lambda_{q_3}^{(1,4)}$

Intermediate result of q_4			
A	B	factor	label
1	3	$\varphi(\tau_5) = e_3 \wedge \neg(e_3 \wedge e_9)$	$\lambda(\tau_5) = \lambda_{q_4}^{(1,3)}$
1	4	$\varphi(\tau_{10}) = e_4 \wedge \neg(e_4 \wedge e_9)$	$\lambda(\tau_{10}) = \lambda_{q_4}^{(1,4)}$

Figure 19.4: Query processing of Q_1 with classical lineage construction

Definition 19.4 (λ -labeling of $\mathcal{T}(\varphi^t)$). Let Q be an algebra query and φ^t be a lineage formula of Q , which is represented by a formula tree $\mathcal{T}(\varphi^t)$. Then, we define a subtree labeling of $\mathcal{T}(\varphi^t)$ as a surjective function $\lambda : \mathcal{T}(\varphi^t) \rightarrow \Lambda$, where Λ denotes the set of all possible labels:

$$\forall \tau \in \mathcal{T}(\varphi^t) : ((\lambda(\tau) = \lambda_q^t) \Leftrightarrow (\varphi(\tau) = \varphi_q^t)).$$

According to Definition (19.4) on Page (216), a subtree τ is labeled by λ_q^t , if and only if τ represents the lineage formula constructed for the intermediate answer tuple t of subquery $q \in \mathcal{SQ}(Q)$.

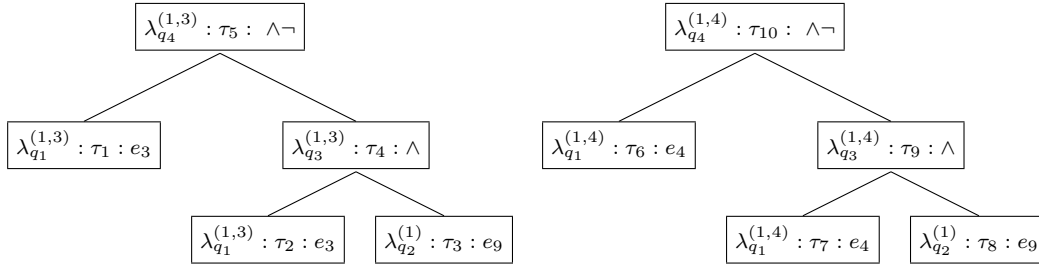
Example 19.7 (Subtree labeling of $\mathcal{T}(\varphi^t)$). In Figure (19.4) on Page (216), we list all labels λ_q^t of Q_1 related to their subtrees. Thereby, we only consider subtrees which are part of our final formula trees.

Please be aware that a specific subtree label $\lambda(\tau) = \lambda_q^t$ already encodes the entire structure of its labeled formula subtree τ . To decode τ , we just need to apply the construction rules of Lemma (6.1) on Page (35) on the given subquery q and return the respective formula tree built for t .

We can express the entire structure of a subtree and its represented factor by means of our simple subtree label λ_q^t .

We make use of this property in the merging phase that follows.

Example 19.8 (Labeled formula tree). The labeled formula trees for $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ are shown in Figure (19.5) on Page (217).

Figure 19.5: Labeled formula trees for $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ with the subtree identifier

For instance, the formula tree of $\varphi_1^{(1,3)}$ contains the subtrees τ_1 and τ_4 annotated with the labels $\lambda(\tau_1) = \lambda_{q_1}^{(1,3)}$ and $\lambda(\tau_4) = \lambda_{q_3}^{(1,3)}$. Both labels express the fact that their corresponding factors

$$\varphi(\tau_1) = e_3 \quad \text{and} \quad \varphi(\tau_4) = e_3 \wedge e_9$$

have been constructed once for the intermediate answer tuple $(1, 3)_{(A, B)}$ of q_1 and q_3 , i.e.,

$$(\lambda(\tau_1) = \lambda_{q_1}^{(1,3)}) \Leftrightarrow (\varphi(\tau_1) = \varphi_{q_1}^{(1,3)}) \quad \text{and} \quad (\lambda(\tau_4) = \lambda_{q_3}^{(1,3)}) \Leftrightarrow (\varphi(\tau_4) = \varphi_{q_3}^{(1,3)}).$$

We finish our discussion about subtree labeling with a short description of situations when a shared factor can emerge during the construction process.

Remark 19.2 (Types of shared factors). *Interestingly, there are just two different types of shared factors. Both of them can be discovered by our labeling technique:*

- *Type (1): A shared factor is created, when a certain subquery is involved in the overall input query more than once.*
- *Type (2): A shared factor is generated, when an intermediate answer tuple is combined with more than one tuple during a join operation.*

Example 19.9 (Types of shared factors). *To give an example for Type (1), we refer to the factor*

$$\varphi_{q_1}^{(1,3)} = e_3 \quad \text{represented by } \tau_1 \text{ and } \tau_2$$

in the formula tree of $\varphi_1^{(1,3)}$, see Figure (19.3) on Page (214). It occurs twice in the overall lineage formula $\varphi_1^{(1,3)}$, because $q_1 = R_2$ is included twice in Q_1 of Figure (19.2) on Page (213). The subtrees τ_1 and τ_2 are both labeled by $\tau_{q_1}^{(1,3)}$.

In addition, we have the shared factor e_9 as an instance of Type (2) in $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$. It appears twice in the subformulas $(e_3 \wedge e_9)$ and $(e_4 \wedge e_9)$, since tuple t_9 is joined with t_3 and t_4 during the processing subquery $q_3 = R_2 \bowtie R_4$ of Figure (19.2) on Page (213). The respective subtrees τ_3 and τ_8 in Figure (19.3) on Page (214) are identically labeled as

$$\lambda(\tau_3) = \lambda(\tau_8) = \lambda_{q_2}^{(1)},$$

because τ_8 is a multiplied copy of τ_3 .

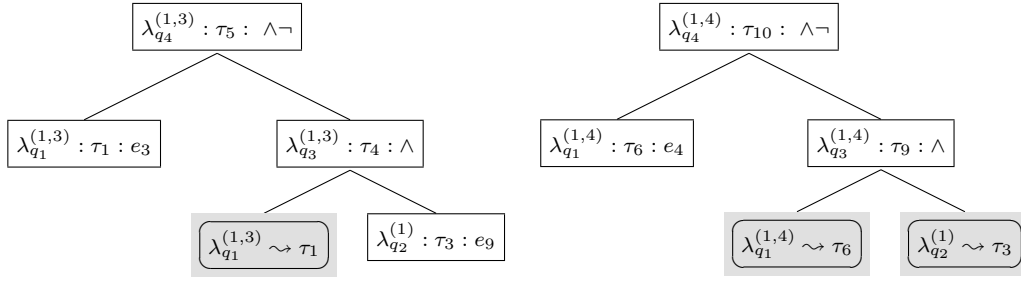


Figure 19.6: Compressed formula trees encoding $\varphi_1^{(1,3)}$ and $\varphi_1^{(1,4)}$ with subtree labels and subtree identifiers

Merging phase

After constructing and labeling formula subtrees in the first two phases, we are now ready to compress our formula trees by eliminating unnecessary subtrees.

We do not have to manage multiple instances of subtrees, if they all encode the same shared factor. Instead, it is sufficient to keep one of them and replace each redundant subtree with a reference pointing to that representative.

More specifically, we look for identical labeled subtrees by comparing subtree pairs. Once we have identified two matching labels, we choose one of the found subtrees and redirect its incoming edge(s) to the root of the remaining subtree.

Example 19.10 (Merging of redundant subtrees). *To demonstrate our final merging phase, we again investigate the formula tree for $\varphi_1^{(1,3)}$ depicted in Figure (19.6) on Page (218). In this formula tree, we have two subtrees τ_1 and τ_2 encoding the same factor $\varphi(\tau_1) = e_3 = \varphi(\tau_2)$. The second subtree τ_2 is redundant, since the structure of the factor e_3 is sufficiently captured by the first subtree τ_1 .*

Thus, we delete τ_2 from the overall formula tree by resetting its incoming edge to the root node of τ_1 as illustrated in Figure (19.6) on Page (218). As a result, subtree τ_2 is not reachable any more.

As exemplified in Figure (19.6) on Page (218), we do not longer deal with classical tree structures after the merging phase. However, every path from the root to a specific leaf is still preserved, if we enumerate the node types on the considered paths. To distinguish our merged formula trees from the classical ones, we call them *compressed formula trees*.

19.3 Optimal λ -labeling

This section studies the query class for which our λ -labeling is *optimal*.

The optimality of our λ -labeling is far from trivial, since we only use simplified labels that can be computed linearly.

Please remember that the generation of the labels of PrDB requires quadratic complexity, since they basically encode the entire syntactical structures of all lineage formulas, see Section (19.1).

First, we recap that our merging phase exploits the following implication

$$\forall \tau, \hat{\tau} \in \mathcal{T}(\varphi^t) : (\lambda(\tau) = \lambda_q^t = \lambda(\hat{\tau})) \Rightarrow (\varphi(\tau) = \varphi(\hat{\tau})).$$

It is derived from Definition (19.4) on Page (216) and states that two subtrees with the same label represent a syntactical identical factor. Is this implication also valid in its opposite direction, i.e.,

$$\forall \tau, \hat{\tau} \in \mathcal{T}(\varphi^t) : (\lambda(\tau) = \lambda_q^t = \lambda(\hat{\tau})) \stackrel{?}{\Leftarrow} (\varphi(\tau) = \varphi(\hat{\tau}))?$$

If it is true, all subtrees that encode syntactical identical factors would have been assigned the same label. Consequently, we would capture all possible merging operations within our merging phase. We consider such a subtree labeling as *optimal*.

Definition 19.5 (Optimal subtree labeling). *Let φ^t be a lineage formula represented by a formula tree $\mathcal{T}(\varphi^t)$. Then, we say that a subtree labeling $\lambda : \mathcal{T}(\varphi^t) \rightarrow \Lambda$ is optimal, if*

$$\forall \tau, \hat{\tau} \in \mathcal{T}(\varphi^t) : (\lambda(\tau) = \lambda_q^t = \lambda(\hat{\tau})) \Leftrightarrow (\varphi(\tau) = \varphi(\hat{\tau}))$$

is given.

Remarkably, our simple labels can ensure an optimal subtree labeling for a very large subset of **SPJUD**-queries, which includes all types of relational operators.

In fact, there is only a small set of queries, which can have different labels for subtrees representing identical factors. For such queries, we could miss a few possible merging operations.

To the contrary, we denote the query class, for which we can guarantee an optimal subtree labeling with **SPJUD**^{*}. Please be aware that **SPJUD**^{*} is foremost a theoretical tool for the ongoing discussions.

In practice, we do not restrict our query language to queries from **SPJUD**^{*}, since our technique is applicable on *all* **SPJUD**-queries. We have to deal with redundant nodes only in some rare cases, see Section (20.2).

In order to define **SPJUD**^{*}, we exclude all queries, where projection operations with *different* projection attributes are applied on *identical* subqueries.

Definition 19.6 (Query class **SPJUD**^{*}). *Let $\mathcal{SQ}(Q)$ be the set of all subquery identifiers of a given algebra query Q . Then, we define the query class **SPJUD**^{*} as follows:*

$$\mathbf{SPJUD}^* := \{Q \in \mathbf{SPJUD} \mid (\pi_{\mathcal{A}}(q), \pi_{\mathcal{B}}(q), q \in \mathcal{SQ}(Q)) \Rightarrow (\mathcal{A} = \mathcal{B})\}.$$

The query class **SPJUD**^{*} is a strict subset of **SPJUD**. However, it still encompasses a large superset of *all* non-repeating queries:

$$\mathbf{nrSPJ} \subset \mathbf{nrSPJUD} \subset \mathbf{SPJUD}^* \subset \mathbf{SPJUD}.$$

Algorithm 7: `insertPaths($d.e, Q, node$)`

```

1   $t := d_{\text{vars}(\text{head}(Q))};$ 
2   $q := Q;$ 
3  if  $\lambda_q^t \in \text{getLabels}(\text{Fa}(q))$  then
4     $edges := \text{getIncomingEdges}(node);$ 
5     $\text{setEdgesToNode}(edges, \text{getLinkedNode}(\text{Fa}(q), \lambda_q^t));$ 
6    return;
7  else
8     $\text{insertFactor}(\text{Fa}(q), \lambda_q^t, node);$ 
9  end
10 switch  $Q$  do
11   case  $Q = \pi_A(Q_1)$ 
12      $node := \boxed{\vee};$ 
13      $key := d_{\text{vars}(\text{head}(Q_1) \setminus A)};$ 
14      $\text{insertPaths}(d.e, Q_1, \text{createIfAbsent}(node.children[key]));$ 
15      $\vdots$ 
16   case  $Q = R$ 
17      $node := \boxed{d.e};$ 
18   endsw
19 endsw

```

Example 19.11 (Query class **SPJUD***). All queries discussed so far in this thesis belong to **SPJUD***. That is, we can build compressed formula trees for each of them without having any redundant nodes.

In Section (20.2), we present the query $Q_2 = \pi_\emptyset(\pi_A(\hat{Q}) \bowtie \pi_B(\hat{Q}))$ as a counterexample to **SPJUD***.

19.4 Extended vertical lineage construction

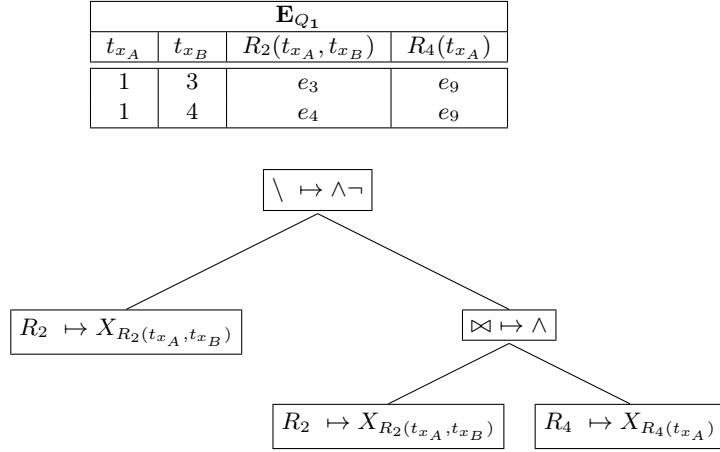
Next, we enhance the basic version of our **vlc**-algorithm (Chapter (15)) by means of our newly introduced factorization and compression technique. Instead of classical formula trees, the augmented **vlc**-algorithm produces compressed formula trees.

Please recall the nature of the three consecutive phases our factorization and compression approach consists of, namely:

- the constructing of formula trees,
- the labeling of formula subtrees, and
- the merging of redundant factors.

The labeling and merging phases are performed in a top-down fashion, i.e., we label and merge always from the root to the leaves. In contrast to the classical construction rules, the **vlc**-algorithm also works in this direction. This facilitates a very easy integration of the labeling and merging phase into our construction algorithm.

Our extended **vlc**-algorithm can conveniently unify all three phases in just *one* processing step per node. This provides on-the-fly construction and compression.

Figure 19.7: Event relation \mathbf{E}_{Q_1} and the implicit transformation mapping based on Q_1

$\text{Fa}(q_1)$	
label	link to
$\lambda_{q_1}^{(1,3)}$	root(τ_1)
$\lambda_{q_1}^{(1,4)}$	root(τ_6)

$\text{Fa}(q_2)$	
label	link to
$\lambda_{q_2}^{(1)}$	root(τ_3)

$\text{Fa}(q_3)$	
label	link to
$\lambda_{q_3}^{(1,3)}$	root(τ_4)
$\lambda_{q_3}^{(1,4)}$	root(τ_9)

$\text{Fa}(q_4)$	
label	link to
$\lambda_{q_4}^{(1,3)}$	root(τ_5)
$\lambda_{q_4}^{(1,4)}$	root(τ_{10})

Figure 19.8: Factor catalogs for Q_1

To be more specific, we only need to conduct an additional existence test, whenever a specific node is visited. This test checks whether the current node is the root of an already built factor. For this purpose, we exploit our subtree labels introduced in Section (19.2). We only have to look for identical subtree labels λ_q^t instead of comparing formula subtrees node-by-node. If a tested factor already exists, we stop the vertical construction of the current path by simply redirecting all incoming edges of the current node to the root node of the already existing subtree.

In practice, we make use of a set of map data structures called *factor catalogs* denoted as $\text{Fa}(q)$. They are defined for each subquery q of $\mathcal{SQ}(Q)$ and store all subtree labels λ_q^t referring to the subquery q . In addition to the subtree labels, each entry of $\text{Fa}(q)$ also contains a direct link to the corresponding root node of $\lambda(\tau) = \lambda_q^t$. By using the factor catalogs, we can seamlessly integrate our entire factorization and compression mechanism into the original **vlc**-algorithm.

In detail, the **vlc**-algorithm implements the required subtree existence test by only one look up operation in $\text{Fa}(q)$, which checks whether a given subtree label is already registered, see Line (3) in Algorithm (7) on Page (220). If a subtree label is already stored in $\text{Fa}(q)$, we retrieve the respective linked root node as the destination for the following merging operation of Line (4) and (5). Otherwise, we create a new label for the unregistered subtree and insert it with a link to the current node into $\text{Fa}(q)$, see Line (8).

Example 19.12 (Combination of **vlc**-algorithm with lineage factorization and compression). In Example (19.3) on Page (212), we already built two lineage formulas

$$\varphi_1^{(1,3)} = e_3 \wedge \neg(e_3 \wedge e_9) \quad \text{and} \quad \varphi_1^{(1,4)} = e_4 \wedge \neg(e_4 \wedge e_9)$$

of Q_1 by the classical construction rules of Lemma (6.1) on Page (35). The two built formula trees are shown in Figure (19.3) on Page (214).

As an alternative to the formula trees of Figure (19.3) on Page (214), our extended **vlc**-algorithm construct the two compressed formula trees depicted in Figure (19.6) on Page (218).

The corresponding event relation \mathbf{E}_{Q_1} and the implicit transformation mapping between algebra operators, the relevant condition and the atomic tuple events are shown in Figure (19.7) on Page (221). Furthermore, we provide in Figure (19.8) on Page (221) the generated factor catalogs for our query Q_1 .

Finally, we confirm that the asymptotic complexity of our algorithm does not change, if we implement our label test using an appropriate hash function. The maximal number of hash values is bounded by the maximal lengths of our lineage formula (Lemma (6.2) on Page (37)), since we maximally store one factor catalog entry per formula tree node.

Lemma 19.1 (Complexity of enhanced **vlc**-algorithm). *Let Q be an algebra query with its domain calculus query*

$$Q^c = \{t \mid \Phi_{\langle \bar{x} \mid \mathbf{D} \rangle}^t\} \equiv \{t \mid \bigvee_{d \in \mathbf{D}} \phi_{\langle \bar{x} \mid d \rangle}^t\}.$$

*It is applied on a probabilistic database **pdb** consisting of all possible tuple combinations of $R_1(W^{max}), \dots, R_m(W^{max})$. Then, the number of steps needed for determining the query result $Q(\mathbf{pdb})$ is bounded by*

$$|Q|^2 * \max(|R_1(W^{max})|, \dots, |R_m(W^{max})|)^{|Q|},$$

*if we use our augmented **vlc**-algorithm of Algorithm (7) on Page (220) and implement all factor catalogs as hash maps.*

Proof. In addition to Lemma (16.5) on Page (169), we only conduct one constant look-up operation for the new factor existence test in Line (3) of Algorithm (7) on Page (220). \square

19.5 Experiments: compressed data structures

In our final series of experiments, we compared the construction of uncompressed and compressed data structures for representing lineage formulas.

In contrast to our former experiments, we only explored a selection of three basic approaches:

- **share plans**: nested lineage formulas built by means of share plans as used in SPROUT2,
- **bisimulation**: factor networks compressed by a bisimulation algorithm as developed for PrDB, and
- **λ -labeling**: factor catalogs generated by vertical lineage construction as devised for Prophecy.

We next present:

- notable implementation details for the three basic approaches under investigation,
- the data we measured, and
- our experimental observations.

Lineage factorization and compression: share plans

For factorizing and compressing nested lineage formulas, we employed share plans proposed by Olteanu and Wen [84]. Please be aware that share plans become safe plans, if we consider tractable queries [84]. Therefore, we could directly use the safe forms of our example queries of Figure (A.5) and (A.6) on Page (250) and (251) in order to implement share plans.

Lineage factorization and compression: bisimulation

Lineage factorization and compression are two key features of the PrDB systems. Sen, Deshpande, and Getoor devised in [101] a bisimulation algorithm for reducing a given factor network. Roughly speaking, their method recursively computes and compares the inputs and types of all factors in order to identify shared factors, see [101] and Section (19.1).

In our experiments, we studied a respective bisimulation algorithm for factorizing and compressing lineage formulas, which worked on the factor networks produced in the relational processing and lineage construction phase discussed in Section (13.5) and (15.2).

Lineage factorization and compression: λ -labeling

Our λ -labeling technique for lineage factorization and compression is intensively laid out in first sections of this chapter. In Algorithm (7) on Page (220), we directly integrated the three phases of our λ -labeling approach into our vertical lineage construction algorithm. Since we only built merged formula trees, there was no need for a dedicated compression step.

Measured data

For our fourth series of experiments, we provide all measured properties and computation times of an investigated approach in a compressed and uncompressed variant, see columns *uncompr* and *compr* in Figure (19.9) and (19.10) on page (224) and (225). In particular, we focused on following data values:

- the accumulated number of all formula nodes per approach as given in the rows labeled with *lineage nodes* of Figure (19.9) and (19.10),
- the accumulated number of all steps for optimizing and evaluating lineage formulas per approach, see rows labeled with *opt/prob steps* of Figure (19.9) and (19.10), and
- the mean of the processing times (wall-clock) of 100 runs for compressing all lineage formulas within our probabilistic query engine per approach, see rows labeled with *compression* in Figure (19.9) and (19.10).

As before, we also provide the times of all our previous processing phases and calculate a total time.

Experimental observations

Finally, we discuss the properties and computation times obtained for exploring uncompressed and compressed versions of all the tested approaches.

Measured properties: compressed and uncompressed lineage nodes and probability computation computation steps

- **share plans:** The number of lineage nodes and probability computation steps could be reduced for the compressed variant of share plans.
- **bisimulation:** Please be aware that factor networks were already simplified by shared factors of Type (2) (Remark (19.2) on Page (217)) in their uncompressed version. Shared factors of Type (1) were not observed in the tested queries. Therefore, the numbers of lineage nodes were identical in both versions.
- **λ -labeling:** For its compressed variant, our method showed the same lineage node count already known from the compressed version of the former two techniques. In essence, all our investigated techniques were able to identify the same amount of shared factors, since there was a lack of shared factors of Type (1) in our explored queries.

Measured properties: lineage compression							
query	property	nested lineage / share plans (SPROUT2)		networks / bisimulation (PrDB)		vertical / λ -labeling (Prophecy)	
		uncompr.	compr.	uncompr.	compr.	uncompr.	compr.
IMDB-Q1	RDBMS relation(s)	1	1	13	13	7	7
	RDBMS tuples	5,473	5,473	369,212	369,212	5,480	5,480
	lineage nodes	372,678	104,212	104,212	104,212	372,678	104,212
	opt/prob steps	134,223	90,464	90,464	90,464	134,223	90,464
IMDB-Q2	RDBMS relation(s)	1	1	7	7	5	5
	RDBMS tuples	30	30	37,298	37,298	3,400	3,400
	lineage nodes	408,280	35,298	35,298	35,298	408,280	35,298
	opt/prob steps	102,175	30,055	30,055	30,055	102,175	30,055
IMDB-Q3	RDBMS relation(s)	1	1	8	8	5	5
	RDBMS tuples	1,516	1,516	940,008	940,008	1,516	1,520
	lineage nodes	296,748	80,636	80,636	80,636	296,748	80,636
	opt/prob steps	81,009	73,429	73,429	73,429	81,009	81,009
IMDB-Q4	RDBMS relation(s)	1	-	10	10	1	1
	RDBMS tuples	2,216	-	926,081	926,081	2,995	2,995
	lineage nodes	38,572	-	26,382	26,382	38,572	26,382
	opt/prob steps	14,075	-	9,643	9,643	14,075	9,643
IMDB-Q5	RDBMS relation(s)	1	-	12	12	1	1
	RDBMS tuples	5,535	-	1,744,059	1,744,059	6,257	6,257
	lineage nodes	139,188	-	98,151	98,151	139,188	98,151
	opt/prob steps	52,189	-	37,916	37,916	52,189	37,916
IMDB-Q6	RDBMS relation(s)	1	-	12	12	1	1
	RDBMS tuples	141	-	655,243	655,243	1,480	1,480
	lineage nodes	5,518	-	4,356	4,356	5,518	4,356
	opt/prob steps	2,173	-	1,489	1,489	2,173	1,489

Measured computation times: lineage compression							
query	part	nested lineage / share plans (SPROUT2)		networks / bisimulation (PrDB)		vertical / λ -labeling (Prophecy)	
		uncompr.	compr.	uncompr.	compr.	uncompr.	compr.
IMDB-Q1	relational	16.806	16.856	3.301	3.288	1.379	1.385
	lineage	0.707	0.71	2.342	2.337	1.56	1.01
	compression	-	0.13	-	1.039	-	0.15
	probability	0.122	0.079	0.081	0.08	0.12	0.079
	total	17.635	17.375	5.724	6.744	3.059	2.624
IMDB-Q2	relational	17.971	18.001	0.368	0.37	0.267	0.262
	lineage	0.532	0.529	0.171	0.175	0.2	0.19
	compression	-	0.11	-	1.61	-	0.029
	probability	0.09	0.004	0.005	0.004	0.089	0.006
	total	18.593	18.534	0.544	0.705	0.556	0.487
IMDB-Q3	relational	190.858	190.858	7.075	7.074	2.839	2.838
	lineage	0.359	0.359	9.289	9.29	0.297	0.209
	compression	-	0.08	-	2.06	-	0.039
	probability	0.074	0.023	0.024	0.025	0.075	0.023
	total	191.291	191.32	16.388	18.449	3.211	3.109
IMDB-Q4	relational	14.77	-	3.699	3.698	0.097	0.095
	lineage	0.048	-	4.834	4.831	0.336	0.301
	compression	-	-	-	3.228	-	0.01
	probability	0.016	-	0.011	0.012	0.016	0.01
	total	14.833	-	8.544	11.769	0.449	0.416
IMDB-Q5	relational	52.796	-	15.478	15.477	0.593	0.595
	lineage	0.201	-	64.724	63.994	1.578	0.788
	compression	-	-	-	44.25	-	0.029
	probability	0.043	-	0.029	0.03	0.041	0.029
	total	53.041	-	80.231	123.751	2.212	1.441
IMDB-Q6	relational	19.746	-	5.375	5.37	0.455	0.453
	lineage	0.008	-	4.227	4.23	0.037	0.02
	compression	-	-	-	3.819	-	0.004
	probability	0.002	-	0.002	0.001	0.002	0.001
	total	19.757	-	9.604	13.42	0.494	0.478

Figure 19.9: IMDB queries: measured properties for lineage compression within probabilistic query engine

Measured properties: lineage compression							
query	property	nested lineage / share plans (SPROUT2)		networks / bisimulation (PrDB)		vertical / λ -labeling (Prophecy)	
		uncompr.	compr.	uncompr.	compr.	uncompr.	compr.
TPCH-Q1	RDBMS relation(s)	1	1	6	6	1	1
	RDBMS tuples	125	125	154,047	154,047	1,725	1,725
	lineage nodes	4,718	2,014	2,014	2,014	4,718	2,014
	opt/prob steps	1,617	1,117	1,117	1,117	1,617	1,117
TPCH-Q2	RDBMS relation(s)	1	1	10	10	1	1
	RDBMS tuples	961	961	176,761	176,761	14,752	14,752
	lineage nodes	43,054	31,234	31,234	31,234	43,054	31,234
	opt/prob steps	17,116	10,601	10,601	10,601	17,116	10,601
TPCH-Q3	RDBMS relation(s)	1	1	12	12	6	6
	RDBMS tuples	38	38	239,647	239,647	11,453	11,453
	lineage nodes	72,488	60,904	60,904	60,904	72,488	60,904
	opt/prob steps	22,527	20,058	20,058	20,058	22,527	20,058
TPCH-Q4	RDBMS relation(s)	1	1	11	11	6	6
	RDBMS tuples	14	14	361,648	361,648	1,125	1,125
	lineage nodes	159,204	99,238	99,238	99,238	159,204	99,238
	opt/prob steps	43,210	37,232	37,232	37,232	43,210	37,232
TPCH-Q5	RDBMS relation(s)	1	-	11	11	6	6
	RDBMS tuples	99	-	151,520	151,520	1,520	1,520
	lineage nodes	271,788	-	51,788	51,788	271,788	51,788
	opt/prob steps	68,343	-	50,233	50,233	68,343	50,233
TPCH-Q6	RDBMS relation(s)	1	-	11	11	6	6
	RDBMS tuples	63	-	348,128	348,128	5,045	5,045
	lineage nodes	233,362	-	128,974	128,974	233,362	128,974
	opt/prob steps	68,704	-	61,001	61,001	68,704	61,001
TPCH-Q7	RDBMS relation(s)	1	-	14	14	7	7
	RDBMS tuples	27	-	493,531	493,531	2,166	2,166
	lineage nodes	100,180	-	66,233	66,233	100,180	66,233
	opt/prob steps	29,483	-	28,456	28,456	29,483	28,456

Measured computation times: lineage compression							
query	part	nested lineage / share plans (SPROUT2)		networks / bisimulation (PrDB)		vertical / λ -labeling (Prophecy)	
		uncompr.	compr.	uncompr.	compr.	uncompr.	compr.
TPCH-Q1	relational	15.082	15.051	2.173	2.176	0.113	0.115
	lineage	0.036	0.038	3.119	3.114	0.144	0.082
	compression	-	0.01	-	1.58	-	0.005
	opt/prob	0.007	0.001	0.002	0.002	0.007	0.001
	total	15.125	15.1	5.294	6.872	0.264	0.203
TPCH-Q2	relational	10.96	10.909	2.353	2.403	0.488	0.49
	lineage	0.126	0.128	1.733	1.759	0.946	0.183
	compression	-	0.017	-	0.523	-	0.031
	opt/prob	0.016	0.012	0.013	0.012	0.015	0.012
	total	11.102	11.066	4.099	4.697	1.449	0.716
TPCH-Q3	relational	12.847	12.799	3.721	3.655	1.683	1.659
	lineage	0.109	0.111	1.891	1.881	0.542	0.49
	compression	-	0.03	-	0.745	-	0.041
	opt/prob	0.024	0.015	0.017	0.016	0.024	0.016
	total	12.98	12.955	5.629	6.297	2.249	2.206
TPCH-Q4	relational	55.117	55.102	5.88	5.98	1.207	1.182
	lineage	0.324	0.337	5.535	5.198	0.186	0.141
	compression	-	0.051	-	1.155	-	0.029
	opt/prob	0.045	0.02	0.01	0.01	0.045	0.01
	total	55.487	55.969	11.425	12.343	1.438	1.362
TPCH-Q5	relational	4.674	-	0.81	0.82	0.37	0.38
	lineage	0.438	-	1.009	0.984	0.113	0.062
	compression	-	-	-	0.514	-	0.099
	opt/prob	0.056	-	0.01	0.01	0.056	0.01
	total	5.167	-	1.829	2.328	0.539	0.551
TPCH-Q6	relational	12.307	-	3.65	3.611	0.898	0.911
	lineage	0.278	-	1.387	1.404	0.498	0.301
	compression	-	-	-	1.549	-	0.059
	opt/prob	0.054	-	0.019	0.02	0.055	0.019
	total	12.639	-	5.056	6.584	1.451	1.2
TPCH-Q7	relational	15.386	-	3.962	4.001	1.381	1.384
	lineage	0.122	-	1.979	1.997	0.347	0.186
	compression	-	-	-	5.662	-	0.12
	opt/prob	0.098	-	0.058	0.056	0.099	0.059
	total	15.606	-	5.999	11.716	1.827	1.749

Figure 19.10: TPC-H queries: measured properties and computation times for lineage compression within probabilistic query engine

Measured computation times: lineage compression within the probabilistic query engine

- **share plans:** The costs of compression were minimal, since it only involved a single merging run over the already built formula trees. The constant label creation has been already be done on query level.
- **bisimulation:** In contrast to the other both methods, the labeling of all factors using bisimulation required the largest amount of time, since the label for a single factor had to be computed over all labels determined for its operands.
- **λ -labeling:** Despite the generation and testing of labels have been directly integrated into our **vlc**-algorithm, we specifically provide the required times for the corresponding computation and look up times of all key values. In total, we observed faster lineage construction times, since all built lineage formulas were already compressed.

19.6 Summary

In this chapter, we presented our third practical contribution of this thesis. First, we introduced the core ideas of our λ -labeling approach in the context of the classical construction rules. Afterwards, they were seamlessly incorporated in our vertical lineage construction algorithm.

Furthermore, we discussed the optimality of our λ -labeling and defined the query class **SPJUD***, for which no redundant nodes are constructed. Finally, the measured properties and processing times of our last series of experiments were presented.

In essence, we showed that how our λ -labeling approach is capable of fulfilling the stated design goals:

- **Design goal (1): full relational algebra support:** the subquery identifiers of our λ -labeling support all relational standard operators,
- **Design goal (2): finding all shared factors (query class):** there are no redundant subtrees built for the query class **SPJUD***,
- **Design goal (3): efficient label computation:** the generation of a single subtree label, which involves a subquery and a tuple identifier is constant, and
- **Design goal (4): on-the-fly factorization:** the direct integration of our labeling and merging phase into the **vlc**-algorithm only constructs already compressed formula trees.

Chapter 20

Advanced aspects of lineage factorization and compression

In this chapter, we present the following themes:

- Section (20.1): the correctness of our lineage factorization and compression method and
- Section (20.2): the optimality of our λ -labeling for the query class **SPJUD**^{*}.

20.1 Correctness of λ -labeling approach

The main goal of this section is to show that classical formula trees can always be replaced by our compressed counterparts. In order to support the following Lemma (20.1) on Page (227), we give in Algorithm (8) on Page (228) a simple algorithm, which traverses all nodes of a given classical or compressed formula tree. It takes a root node of a classical or compressed formula tree and returns the represented lineage formula.

Example 20.1 (Traverse paths). *When we apply our algorithm **traversePaths** on the root nodes of the two uncompressed and two compressed formula trees discussed in Example (19.3) and (19.6) on Page (214) and (218), we obtain the same pair of lineage formulas in both cases:*

$$\begin{aligned}\mathbf{traversePaths}(\mathit{root}(\tau_5)) &= (e_3 \wedge \neg(e_3 \wedge e_9)) \\ \mathbf{traversePaths}(\mathit{root}(\tau_{10})) &= (e_4 \wedge \neg(e_4 \wedge e_9)).\end{aligned}$$

Instead of just returning the encoded lineage formula, we could also base more complex algorithms on a compressed formula tree, e.g., algorithms for calculating the probability $\mathbf{P}(\varphi^t)$. However, our auxiliary algorithm helps us first and foremost to prove the congruent traversing of all paths between a classical formula tree and its compressed version.

Lemma 20.1 (Correctness of lineage compression). *Let τ be a classical formula tree and $\hat{\tau}$ be its compressed version. When we consider the two lineage formulas returned by Algorithm (8) on Page (228):*

$$\varphi(\tau) := \mathbf{traversePaths}(\mathit{root}(\tau)) \quad \text{and} \quad \hat{\varphi}(\hat{\tau}) := \mathbf{traversePaths}(\mathit{root}(\hat{\tau})),$$

then

$$\varphi(\tau) = \hat{\varphi}(\hat{\tau})$$

holds.

Algorithm 8: $\text{traversePaths}(\text{node})$

```

1 switch  $\text{node.type}$  do
2   case  $\text{node.type} \in \{\Box\}$ 
3      $\varphi(\tau) := \text{node.type} + \text{'('}$ ;
4     foreach  $\text{childNode} \in \text{node.children}$  do
5        $\varphi(\tau) := \varphi(\tau) + \text{traversePaths}(\text{childNode}) + \text{'}, \text{'}$ ;
6     end
7     return  $\varphi(\tau) + \text{'F'}$ ;
8   case  $\text{node.type} \in \{\Box\wedge, \Box\vee, \Box\neg\wedge\}$ 
9      $\varphi(\tau) := \text{'('} + \text{traversePaths}(\text{node.left}) + \text{node.type} + \text{traversePaths}(\text{node.right}) + \text{'})'$ ;
10    return  $\varphi(\tau)$ ;
11   case  $\text{node.type} \in \{\Box e_i, \Box T, \Box F\}$ 
12      $\varphi(\tau) := \text{node.type}$ ;
13     return  $\varphi(\tau)$ ;
14   endsw
15 endsw

```

Proof. We prove the proposition by induction over the number of applied merging operations in our compressed formula tree $\hat{\tau}$. Thereby, we derive $\hat{\tau}$ from τ .

I.) Induction basis (compression with $n = 0$ merging operations):

Our merging phase begins with a classical formula tree τ . Obviously, we can be sure that the formula trees τ and $\hat{\tau}$ are identical, if we do not perform any merging operations.

II.) Induction assumption (compression with n merging operations):

The two lineage formulas

$$\varphi(\tau) := \text{traversePaths}(\text{root}(\tau)) \quad \text{and} \quad \hat{\varphi}(\hat{\tau}) := \text{traversePaths}(\text{root}(\hat{\tau}))$$

are identical, i.e., $\varphi(\tau) = \hat{\varphi}(\hat{\tau})$, if the compressed formula tree $\hat{\tau}$ is built by n or less merging operations.

III.) Induction step (compression with n merging operations):

The starting point of our discussion is a formula tree τ and its compressed version $\hat{\tau}$. The compressed formula tree $\hat{\tau}$ is generated by n or less merging operations. Both formula trees are labeled during the labeling phase of Section (19.2).

First of all, we know that each node in τ and $\hat{\tau}$ is the root node of a specific subtree of τ or $\hat{\tau}$. Accordingly, we can address each node in τ or $\hat{\tau}$ as $\text{root}(\tau_i)$ or $\text{root}(\hat{\tau}_i)$, where $\tau_i/\hat{\tau}_i$ is one of the respective subtrees.

Next, we fix two specific subtrees $\hat{\tau}_{m_1}$ and $\hat{\tau}_{m_2}$ of $\hat{\tau}$ with identical labels in order to prepare our $(n+1)$ -th merging operation, i.e.,

$$\lambda(\hat{\tau}_{m_1}) = \lambda_q^t = \lambda(\hat{\tau}_{m_2}).$$

Since $\hat{\tau}$ is built from τ , we can be sure that there are two equally labeled subtrees in τ :

$$\lambda(\tau_{m_1}) = \lambda_q^t = \lambda(\tau_{m_2}).$$

According to Definition (19.4) on Page (216), we know that two classical subtrees, which have the same subtree label represent the same factor. Thanks to our induction assumption, we

can transfer this property to our compressed subtrees $\hat{\tau}_{m_1}$ and $\hat{\tau}_{m_2}$:

$$(\lambda(\tau_{m_1}) = \lambda_q^t = \lambda(\tau_{m_2})) \Rightarrow (\varphi(\tau_{m_1}) = \varphi(\tau_{m_2})) \stackrel{\text{(IA)}}{\Rightarrow} (\hat{\varphi}(\hat{\tau}_{m_1}) = \hat{\varphi}(\hat{\tau}_{m_2})).$$

Consequently, the factors $\hat{\varphi}(\hat{\tau}_{m_1})$ and $\hat{\varphi}(\hat{\tau}_{m_2})$ encoded by $\hat{\tau}_{m_1}$ and $\hat{\tau}_{m_2}$ are identical.

Without loss of generality, we assume that our first subtree $\hat{\tau}_{m_1}$ can be reached from the overall root node $\text{root}(\hat{\tau})$ by following node path:

$$\text{root}(\hat{\tau}) \rightarrow \text{root}(\hat{\tau}_{p_0}) \rightarrow \text{root}(\hat{\tau}_{p_1}) \rightarrow \dots \rightarrow \text{root}(\hat{\tau}_{p_n}) \rightarrow \text{root}(\hat{\tau}_{m_1}).$$

In other words, we traverse the roots of the following subtrees: $\hat{\tau}, \hat{\tau}_{p_0}, \hat{\tau}_{p_1}, \dots, \hat{\tau}_{p_n}, \hat{\tau}_{m_1}$.

Now, we perform the $(n+1)$ -th merging operation by redirecting the incoming edge of the node $\text{root}(\hat{\tau}_{m_1})$ to the node $\text{root}(\hat{\tau}_{m_2})$ and obtain the following adjusted path:

$$\text{root}(\hat{\tau}) \rightarrow \text{root}(\hat{\tau}_{p_0}) \rightarrow \text{root}(\hat{\tau}_{p_1}) \rightarrow \dots \rightarrow \text{root}(\hat{\tau}_{p_n}) \rightarrow \text{root}(\hat{\tau}_{m_2}).$$

The remaining point is to show that the syntax of our final lineage formula $\hat{\varphi}(\hat{\tau})$ has not changed. For this purpose, we investigate the behaviour of our algorithm, when it visits the node $\text{root}(\hat{\tau}_{p_n})$, which is the predecessor of the last path node.

If we consider $\hat{\tau}_{p_n}$ as a separate subtree, our algorithm returns $\hat{\varphi}(\hat{\tau}_{p_n})$ starting from $\text{root}(\hat{\tau}_{p_n})$. After our $(n+1)$ -th merging operation, we do not longer move to the node $\text{root}(\hat{\tau}_{m_1})$ after processing the node $\text{root}(\hat{\tau}_{p_n})$. Instead, we go to the node $\text{root}(\hat{\tau}_{m_2})$ and continue to build $\hat{\varphi}(\hat{\tau}_{m_2})$. In other words, the subformula $\hat{\varphi}(\hat{\tau}_{m_1})$ is replaced by $\hat{\varphi}(\hat{\tau}_{m_2})$ within $\hat{\varphi}(\hat{\tau}_{p_n})$. The syntax of $\hat{\varphi}(\hat{\tau}_{p_n})$ is still preserved, since we already followed that $\hat{\varphi}(\hat{\tau}_{m_1}) = \hat{\varphi}(\hat{\tau}_{m_2})$ holds, see above.

This conclusion can be repeated for $\hat{\tau}_{p_{n-1}}$ and $\hat{\tau}_{p_n}$. Again, we just substitute a subtree with a subtree encoding an identical factor. When we extend our argumentation on the entire path until we reach the overall root $\text{root}(\hat{\tau})$, it is shown that the syntax of $\hat{\varphi}(\hat{\tau})$ has not changed after the $(n+1)$ -th merging operation. \square

20.2 Optimal subtree labeling

In the following, we prove that our λ -labeling is optimal for the query class **SPJUD**^{*}.

Example 20.2 (Query class **SPJUD**^{*}). *We again point out that, except for query Q_2 introduced next, all our queries studied in this thesis belong to **SPJUD**^{*}. Thus, we can build compressed formula trees for each of them without having any redundant nodes.*

*To the contrary, the following example query Q_2 is not a member of **SPJUD**^{*}:*

$$Q_2 = \pi_{\emptyset}(\pi_A(\hat{Q}) \bowtie \pi_B(\hat{Q})),$$

where the subquery identifiers $\mathcal{SQ}(Q_2) = \{q_1, \dots, q_5\}$ of Q_2 refer to

$$\begin{aligned} q_1 &:= \hat{Q} & q_3 &:= \pi_B(\hat{Q}) & q_5 &:= \pi_{\emptyset}(\pi_A(\hat{Q}) \bowtie \pi_B(\hat{Q})). \\ q_2 &:= \pi_A(\hat{Q}) & q_4 &:= \pi_A(\hat{Q}) \bowtie \pi_B(\hat{Q}) \end{aligned}$$

The query Q_2 is not included in **SPJUD**^{*}, because it performs two different projections

$$q_2 := \pi_A(\hat{Q}) \quad \text{and} \quad q_3 := \pi_B(\hat{Q})$$

on the same subquery \hat{Q} .

For the sake of brevity, we do not specify all details of the involved subquery \hat{Q} . Instead, we assume in Figure (20.1) on Page (231) that the resulting tuples of \hat{Q} and their corresponding lineage formulas are already given.

As mentioned earlier, we cannot guarantee an optimal subtree labeling (Definition (19.5) on Page (219)) for Q_2 , since $Q_2 \notin \mathbf{SPJUD}^*$. That is, we could find subtrees with different subtree labels encoding the same shared factor. When we analyze the intermediate results for all given subqueries of Q_2 in Figure (20.1) on Page (231), we indeed recognize two identical subtrees τ_3 and τ_6 , which are annotated with two different labels

$$\lambda(\tau_3) = \lambda_{q_2}^{(1)} \quad \text{and} \quad \lambda(\tau_6) = \lambda_{q_3}^{(3)}.$$

Despite their different labels, they both represent the same factor:

$$\varphi(\tau_3) = (\varphi_1 \odot \varphi_2) = \varphi(\tau_6).$$

This also means that the subqueries q_2 and q_3 can determine identical factors, despite they are neither identical nor equivalent.

Next, let us imagine that a query contains two critical projection operations as demonstrated in Example (20.2) on Page (229). Then, the two corresponding n-ary disjunctions might fail to share a large number of identical disjunctive operands. However, we emphasize that only the root nodes of those problematic subformulas are redundant in one of our compressed formula subtrees. All inner nodes are shared again. As consequence, the total number of unnecessary nodes keeps relatively low.

Example 20.3 (Redundant nodes for queries from \mathbf{SPJUD}^*). *To exemplify our last statement, we depict in Figure (20.2), and (20.3) on Page (232) and (233) one classical and two compressed formula trees, which all are encoding the same lineage formula $\varphi_2^{(0)}$ created in Figure (20.1) on Page (231).*

By comparing both compressed formula trees, we see that only the root nodes of τ_3 and τ_6 for the shared factor

$$\varphi(\tau_3) \equiv (\varphi_1 \odot \varphi_2) \equiv \varphi(\tau_6)$$

and the root nodes of τ_{12} and τ_{15} for

$$\varphi(\tau_{12}) \equiv \varphi(\varphi_3) \equiv \varphi(\tau_{15})$$

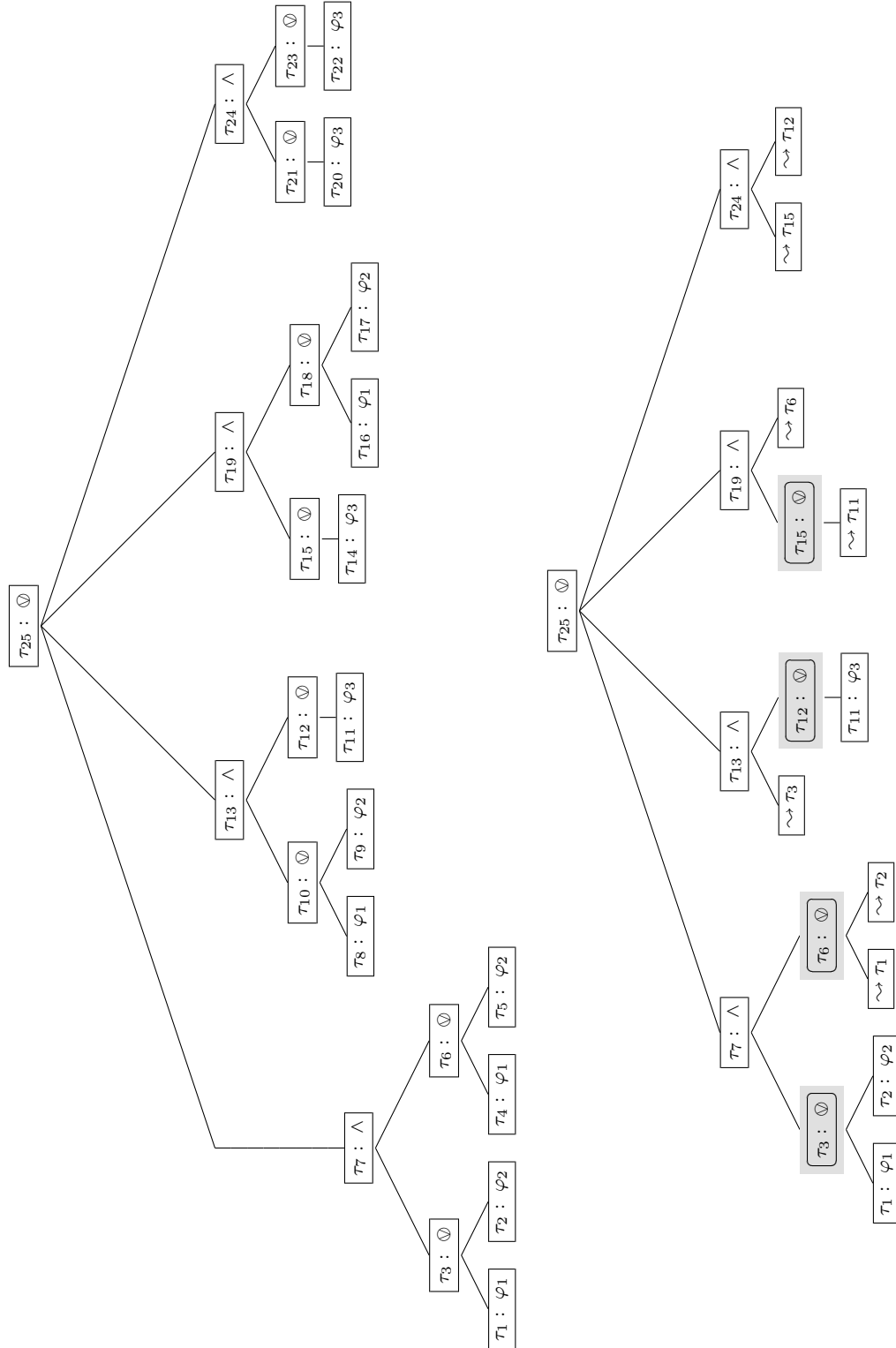
are duplicated.

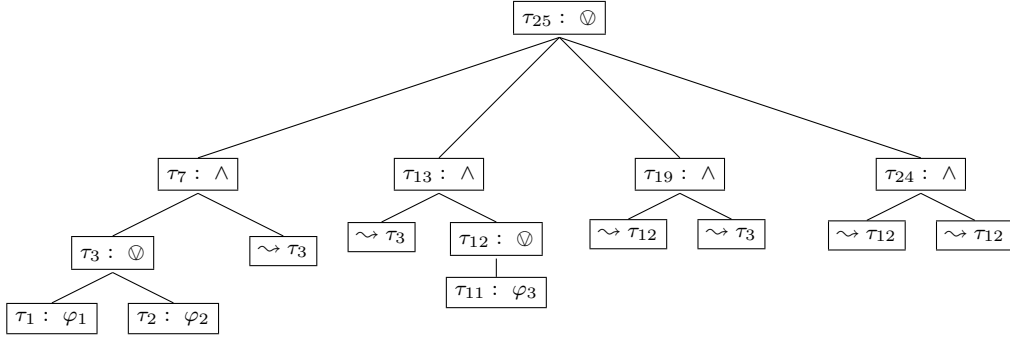
\tilde{Q}				Intermediate result of q_1					Intermediate result of q_2						
A	B	C	φ_Q^t	A	B	C	factor		label		A	factor		label	
1	3	3	φ_1	1	3	3	$\varphi(\tau_1) = \varphi_1$		$\lambda(\tau_1) = \lambda_{q_1}^{(1,3,3)}$		1	$\varphi(\tau_3) = (\varphi_1 \otimes \varphi_2)$		$\lambda(\tau_3) = \lambda_{q_2}^{(1)}$	
1	3	3	φ_2	1	3	3	$\varphi(\tau_2) = \varphi_2$		$\lambda(\tau_2) = \lambda_{q_1}^{(1,3,3)}$		2	$\varphi(\tau_{15}) = \varphi_3$		$\lambda(\tau_{15}) = \lambda_{q_2}^{(2)}$	
2	2	2	φ_3	2	2	2	$\varphi(\tau_{11}) = \varphi_3$		$\lambda(\tau_{11}) = \lambda_{q_1}^{(2,2,2)}$						

Intermediate result of q_3					Intermediate result of q_4				
B	factor			label	A	B	factor		label
3	$\varphi(\tau_6) = (\varphi_1 \otimes \varphi_2)$			$\lambda(\tau_6) = \lambda_{q_3}^{(3)}$	1	3	$\varphi(\tau_7) = ((\varphi_1 \otimes \varphi_2) \wedge (\varphi_1 \otimes \varphi_2))$		$\lambda(\tau_7) = \lambda_{q_4}^{(1,3)}$
					1	2	$\varphi(\tau_{13}) = ((\varphi_1 \otimes \varphi_2) \wedge \varphi_3)$		$\lambda(\tau_{13}) = \lambda_{q_4}^{(1,2)}$
2	$\varphi(\tau_{12}) = \varphi_3$			$\lambda(\tau_{12}) = \lambda_{q_3}^{(2)}$	2	3	$\varphi(\tau_{19}) = (\varphi_3 \wedge (\varphi_1 \otimes \varphi_2))$		$\lambda(\tau_{19}) = \lambda_{q_4}^{(2,3)}$
					2	2	$\varphi(\tau_{24}) = (\varphi_3 \wedge \varphi_3)$		$\lambda(\tau_{24}) = \lambda_{q_4}^{(2,2)}$

Intermediate result of q_5		
	factor	label
()	$\varphi(\tau_{25}) = ((\varphi_1 \otimes \varphi_2) \wedge (\varphi_1 \otimes \varphi_2)) \otimes ((\varphi_1 \otimes \varphi_2) \wedge \varphi_3) \otimes (\varphi_3 \wedge (\varphi_1 \otimes \varphi_2)) \otimes (\varphi_3 \wedge \varphi_3)$	$\lambda_{q_5}^{()}$

Figure 20.1: Query processing of Q_2 with its classical lineage construction


 Figure 20.2: Classical and non-optimal compressed formula trees representing φ_2^{\emptyset}

Figure 20.3: Optimal compressed formula tree encoding $\varphi_2^{()}$

To prove that our λ -labeling is optimal for all queries from **SPJUD**^{*}, we first show that two different answer tuples always have two syntactically different lineage formulas, if we apply the classical construction rule of Lemma (6.1) on Page (35) on an **SPJUD**^{*}-query.

Lemma 20.2 (Syntactically different lineage formulas for distinct tuples). *Let Q be an algebra query from **SPJUD**^{*}. Then,*

$$\forall t, \hat{t} : ((t \neq \hat{t}) \Rightarrow (\varphi_Q^t \neq \varphi_Q^{\hat{t}}))$$

holds, if φ_Q^t and $\varphi_Q^{\hat{t}}$ are constructed by means of Lemma (6.1) on Page (35).

Proof. We verify our proposition by using an induction proof over the number of operators involved in Q .

I.) Induction basis (Q with $n = 1$ operators, i.e., $Q = R$):

According to our relation rule $Q = R$ of Lemma (6.1) on Page (35), we know that all atomic lineage formulas are formulated by *unique* atomic tuple events:

$$\forall t, \hat{t} : ((t \neq \hat{t}) \Rightarrow (\varphi_R^t = e_t \neq e_{\hat{t}} = \varphi_R^{\hat{t}})).$$

II.) Induction assumption (Q with n operators):

Let Q be an algebra query from **SPJUD**^{*}. Then,

$$\forall t, \hat{t} : ((t \neq \hat{t}) \Rightarrow (\varphi_Q^t \neq \varphi_Q^{\hat{t}}))$$

holds, if φ_Q^t and $\varphi_Q^{\hat{t}}$ are constructed by means of Lemma (6.1) on Page (35).

III.) Induction step (Q with $n + 1$ operators):

We consider six cases for the induction step. Each of them stands for one possible $(n + 1)$ -th operator of Q . In all six cases, we can directly use the induction assumption (IA) in a combination with the classical construction rules of Lemma (6.1) on Page (35).

Case (1): $Q = \sigma_F(Q_1)$

$$(t, \hat{t} \in Q) \wedge (t \neq \hat{t}) \Rightarrow (t, \hat{t} \in Q_1) \wedge (t \neq \hat{t})$$

$$\begin{aligned}
&\Rightarrow \varphi_{Q_1}^t \neq \varphi_{Q_1}^{\hat{t}} && \text{(IA)} \\
&\Rightarrow (\varphi_Q^t = \varphi_{Q_1}^t) \wedge (\varphi_Q^{\hat{t}} = \varphi_{Q_1}^{\hat{t}}) && \text{(Lemma (6.1))} \\
&\Rightarrow \varphi_Q^t \neq \varphi_Q^{\hat{t}}
\end{aligned}$$

Case (2): $Q = \pi_{\mathcal{A}}(Q_1)$

$$\begin{aligned}
(t, \hat{t} \in Q) \wedge (t \neq \hat{t}) &\Rightarrow \exists t_1, \hat{t}_1 \in Q_1 : (t_1 = t \bullet t') \wedge (\hat{t}_1 = \hat{t} \bullet \hat{t}') \\
&\Rightarrow t_1 \neq \hat{t}_1 && (t \neq \hat{t}) \\
&\Rightarrow \varphi_{Q_1}^{t_1} \neq \varphi_{Q_1}^{\hat{t}_1} && \text{(IA)} \\
&\Rightarrow (\varphi_Q^t = \dots \otimes \varphi_{Q_1}^{t_1} \otimes \dots) \wedge (\varphi_Q^{\hat{t}} = \dots \otimes \varphi_{Q_1}^{\hat{t}_1} \otimes \dots) && \text{(Lemma (6.1))} \\
&\Rightarrow \varphi_Q^t \neq \varphi_Q^{\hat{t}}
\end{aligned}$$

Case (3): $Q = Q_1 \bowtie Q_2$

$$\begin{aligned}
(t, \hat{t} \in Q) \wedge (t \neq \hat{t}) &\Rightarrow (t_1 \bullet t_2 \in Q_1 \bowtie Q_2) \wedge (\hat{t}_1 \bullet \hat{t}_2 \in Q_1 \bowtie Q_2) \wedge \\
&\quad (t_1 \bullet t_2 \neq \hat{t}_1 \bullet \hat{t}_2) \\
&\Rightarrow (\varphi_{Q_1}^{t_1} \wedge \varphi_{Q_2}^{t_2}) \neq (\varphi_{Q_1}^{\hat{t}_1} \wedge \varphi_{Q_2}^{\hat{t}_2}) && \text{(IA)} \\
&\Rightarrow (\varphi_Q^t = \varphi_{Q_1}^{t_1} \wedge \varphi_{Q_2}^{t_2}) \wedge (\varphi_Q^{\hat{t}} = \varphi_{Q_1}^{\hat{t}_1} \wedge \varphi_{Q_2}^{\hat{t}_2}) && \text{(Lemma (6.1))} \\
&\Rightarrow \varphi_Q^t \neq \varphi_Q^{\hat{t}}
\end{aligned}$$

Case (4): $Q = Q_1 \cup Q_2$

$$\begin{aligned}
(t, \hat{t} \in Q) \wedge (t \neq \hat{t}) &\Rightarrow ((t \in Q_1) \vee (t \in Q_2)) \wedge ((\hat{t} \in Q_1) \vee (\hat{t} \in Q_2)) \wedge (t \neq \hat{t}) \\
&\Rightarrow (\varphi_{Q_1}^t \vee \varphi_{Q_2}^t) \neq (\varphi_{Q_1}^{\hat{t}} \vee \varphi_{Q_2}^{\hat{t}}) && \text{(IA)} \\
&\Rightarrow (\varphi_Q^t = \varphi_{Q_1}^t \vee \varphi_{Q_2}^t) \wedge (\varphi_Q^{\hat{t}} = \varphi_{Q_1}^{\hat{t}} \vee \varphi_{Q_2}^{\hat{t}}) && \text{(Lemma (6.1))} \\
&\Rightarrow \varphi_Q^t \neq \varphi_Q^{\hat{t}}
\end{aligned}$$

Case (5): $Q = Q_1 \setminus Q_2$

$$\begin{aligned}
(t, \hat{t} \in Q) \wedge (t \neq \hat{t}) &\Rightarrow (t, \hat{t} \in Q_1) \wedge (t \neq \hat{t}) \\
&\Rightarrow \varphi_{Q_1}^t \neq \varphi_{Q_1}^{\hat{t}} && \text{(IA)} \\
&\Rightarrow (\varphi_Q^t = \varphi_{Q_1}^t \wedge \neg \varphi_{Q_2}^t) \wedge (\varphi_Q^{\hat{t}} = \varphi_{Q_1}^{\hat{t}} \wedge \neg \varphi_{Q_2}^{\hat{t}}) && \text{(Lemma (6.1))} \\
&\Rightarrow \varphi_Q^t \neq \varphi_Q^{\hat{t}}
\end{aligned}$$

Case (6): $Q = \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(Q_1)$

Analog to Case (1). □

Please note that our last lemma only assures syntactic divergence. The more general implication

$$\forall t, \hat{t} : (t \neq \hat{t}) \stackrel{?}{\Rightarrow} (\varphi_Q^t \neq \varphi_Q^{\hat{t}})$$

is not valid.

Example 20.4 (Equivalent lineage formulas for different tuples). *For the query*

$$Q = Q_1 \setminus Q_1 \quad \text{with} \quad Q_1 \in \mathbf{SPJUD}^*,$$

we can set up two syntactic different lineage formulas for two tuples t, \hat{t} with $t \neq \hat{t}$:

$$(t \neq \hat{t}) \Rightarrow ((\varphi_Q^t = \varphi_{Q_1}^t \wedge \neg(\varphi_{Q_1}^t)) \neq (\varphi_{Q_1}^{\hat{t}} \wedge \neg(\varphi_{Q_1}^{\hat{t}})) = \varphi_{Q_1}^{\hat{t}}),$$

see Case (5) of Lemma (20.2) on Page (233). However, they are equivalent to each other:

$$\varphi_Q^t = \varphi_{Q_1}^t \wedge \neg(\varphi_{Q_1}^t) \equiv F \equiv \varphi_{Q_1}^{\hat{t}} \wedge \neg(\varphi_{Q_1}^{\hat{t}}) = \varphi_{Q_1}^{\hat{t}}.$$

Nonetheless, Lemma (20.2) on Page (233) helps us to formulate our final proof.

Lemma 20.3 (Optimal subtree labeling for \mathbf{SPJUD}^* -queries). *Let $Q \in \mathbf{SPJUD}^*$ be an algebra query with its subquery identifiers $\mathcal{SQ}(Q)$. If $\mathcal{T}(\varphi^t)$ is a formula tree representing a lineage formula φ^t built for Q , then the subtree labeling $\lambda : \mathcal{T}(\varphi^t) \rightarrow \Lambda$ is optimal (Definition (19.5) on Page (219)):*

$$\forall \tau, \hat{\tau} \in \mathcal{T}(\varphi^t) : (\lambda(\tau) = \lambda_q^t = \lambda(\hat{\tau})) \Leftrightarrow (\varphi(\tau) = \varphi(\hat{\tau})).$$

Proof. The (\Rightarrow) -direction of our proposition directly follows from Definition (19.4) on Page (216) that specifies a subtree label. It states that two subtrees annotated by an identical label λ_q^t encode the lineage formula constructed for the intermediate answer tuple t of subquery q :

$$\forall t, \hat{t} : ((\lambda(\tau) = \lambda_q^t = \lambda(\hat{\tau})) \Rightarrow (\varphi(\tau) = \varphi_q^t = \varphi(\hat{\tau}))).$$

In order to show the (\Leftarrow) -direction of our proposition, we prove the equivalent implication:

$$\forall t, \hat{t} : \forall q, \hat{q} \in \mathcal{SQ}(Q) : ((\lambda(\tau) = \lambda_q^t \neq \lambda_{\hat{q}}^{\hat{t}} = \lambda(\hat{\tau})) \Rightarrow (\varphi(\tau) = \varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}} = \varphi(\hat{\tau}))).$$

In other words, we verify that two differently labeled subtrees always encode two syntactically distinct factors. The two subtree labels λ_q^t and $\lambda_{\hat{q}}^{\hat{t}}$ are different, if their defining queries q, \hat{q} and/or the considered tuples t, \hat{t} do not match:

$$\forall t, \hat{t} : \forall q, \hat{q} \in \mathcal{SQ}(Q) : ((\lambda_q^t \neq \lambda_{\hat{q}}^{\hat{t}}) \Rightarrow ((q \neq \hat{q}) \vee (t \neq \hat{t}))).$$

That is, we can derive two possible cases with regards to q and \hat{q} . First, we presume q equals \hat{q} . Then, we directly conclude from Lemma (20.2) on Page (233) that φ_q^t differs from $\varphi_{\hat{q}}^{\hat{t}}$:

$$\forall t, \hat{t} : \forall q, \hat{q} \in \mathcal{SQ}(Q) : (((\lambda_q^t \neq \lambda_{\hat{q}}^{\hat{t}}) \wedge (q = \hat{q}) \wedge (t \neq \hat{t})) \Rightarrow (\varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}})).$$

To prove the second case

$$\forall t, \hat{t} : \forall q, \hat{q} \in \mathcal{SQ}(Q) : (((\lambda_q^t \neq \lambda_{\hat{q}}^{\hat{t}}) \wedge (q \neq \hat{q})) \Rightarrow (\varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}})),$$

we employ an induction proof over the total number of operators n involved in q and \hat{q} .

I.) Induction basis ($n = 2$ overall operators in q and \hat{q}):

The number of operators $n = 2$ in conjunction with $q \neq \hat{q}$ leads to

$$q = R \quad \neq \quad \hat{R} = \hat{q}.$$

Because R and \hat{R} are two different relations, we know by construction (see Lemma (6.1) on Page (35)) that all atomic lineage formulas are given by *unique* atomic tuple events, i.e.,

$$(R \neq \hat{R}) \Rightarrow (\varphi_R^t = e_t \neq e_{\hat{t}} = \varphi_{\hat{R}}^{\hat{t}}).$$

II.) Induction assumption (n operators in q and \hat{q}):

The property

$$\forall t, \hat{t} : ((\lambda_q^t \neq \lambda_{\hat{q}}^{\hat{t}}) \wedge (q \neq \hat{q})) \Rightarrow (\varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}})$$

holds for two subqueries q and \hat{q} of $\mathcal{SQ}(Q)$, if they involve n operators in total.

III.) Induction step ($n + 1$ operations in q and \hat{q}):

In our induction step, we address all possible combinations of q and \hat{q} with

$$q, \hat{q} \in \{R, \quad \sigma_F(q_1), \quad \pi_{\mathcal{A}}(q_1), \quad q_1 \bowtie q_2, \quad q_1 \cup q_2, \quad q_1 \setminus q_2, \quad \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(q_1)\}$$

and $q \neq \hat{q}$ (except for $q = R$ and $\hat{q} = \hat{R}$, which were already treated in our induction basis).

For the sake of compactness, we reduce the number of query combinations to three (generic) cases. To do so, we first conclude that we can omit all query pairs, where q and \hat{q} are out of

$$\{R, \quad \sigma_F(q_1), \quad \rho_{(\mathcal{B} \leftarrow \mathcal{A})}(q_1)\},$$

since these queries produce the same lineage formulas as the query $\pi_{\text{head}(q_1)}(q_1)$, which is covered by our further cases. Then, the query pairs to prove are built from:

$$q, \hat{q} \in \{\pi_{\mathcal{A}}(q_1), \quad q_1 \bowtie q_2, \quad q_1 \cup q_2, \quad q_1 \setminus q_2\}.$$

Case (1): $q = \pi_{\mathcal{A}}(q_1)$ and $\hat{q} = \pi_{\mathcal{B}}(\hat{q}_1)$

In Case (1) and (2), we presume that the attribute sets \mathcal{A} and \mathcal{B} of all considered projection operations are always strict subsets of the head attributes of the underlying subquery, i.e.,

$$\mathcal{A} \subset \text{head}(q_1) \quad \text{and} \quad \mathcal{B} \subset \text{head}(\hat{q}_1).$$

Otherwise, we can directly apply our induction assumption:

$$((q = \pi_{\mathcal{A}}(q_1) = \pi_{\text{head}(q_1)}(q_1) = q_1) \vee (\hat{q} = \pi_{\mathcal{A}}(\hat{q}_1) = \pi_{\text{head}(\hat{q}_1)}(\hat{q}_1) = \hat{q}_1)) \stackrel{(\text{IA})}{\Rightarrow} (\varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}}).$$

Concretely, we show our claimed proposition

$$((q = \pi_{\mathcal{A}}(q_1)) \wedge (\hat{q} = \pi_{\mathcal{B}}(\hat{q}_1))) \Rightarrow (\varphi_q^t = \bigoplus_{(t_1 \in q_1, t_{1,\mathcal{A}}=t)} \varphi_{q_1}^{t_1} \neq \bigoplus_{(\hat{t}_1 \in \hat{q}_1, \hat{t}_{1,\mathcal{B}}=\hat{t})} \varphi_{\hat{q}_1}^{\hat{t}_1} = \varphi_{\hat{q}}^{\hat{t}})$$

by contradicting its negation. So, let us assume that

$$(\pi_{\mathcal{A}}(q_1) \neq \pi_{\mathcal{B}}(\hat{q}_1)) \wedge (\varphi_q^t = \bigoplus_{(t_1 \in q_1, t_{1,\mathcal{A}}=t)} \varphi_{q_1}^{t_1} = \bigoplus_{(\hat{t}_1 \in \hat{q}_1, \hat{t}_{1,\mathcal{B}}=\hat{t})} \varphi_{\hat{q}_1}^{\hat{t}_1} = \varphi_{\hat{q}}^{\hat{t}})$$

is given. In that case, we have pairs of identical lineage subformulas with $\varphi_{q_1}^{t_1} = \varphi_{\hat{q}_1}^{\hat{t}_1}$. Otherwise, the combined lineage formulas differ from each other, i.e., $\varphi_q^t \neq \varphi_{\hat{q}}^{\hat{t}}$.

If we rewrite our induction assumption from

$$(q_1 \neq \hat{q}_1) \Rightarrow (\varphi_{q_1}^t \neq \varphi_{\hat{q}_1}^{\hat{t}}) \quad \text{to} \quad (\varphi_{q_1}^t = \varphi_{\hat{q}_1}^{\hat{t}}) \Rightarrow (q_1 = \hat{q}_1),$$

we can conclude $q_1 = \hat{q}_1$ that, if $\varphi_{q_1}^{t_1} = \varphi_{\hat{q}_1}^{\hat{t}_1}$ holds for all pairs. That leads to the following contradiction:

$$((q \neq \hat{q}) \wedge (q_1 = \hat{q}_1)) \Rightarrow ((\pi_{\mathcal{A}}(q_1) \neq \pi_{\mathcal{B}}(\hat{q}_1)) \wedge (q_1 = \hat{q}_1)) \Rightarrow (\mathcal{A} \neq \mathcal{B}) \not\vdash.$$

Please recall that $(\mathcal{A} \neq \mathcal{B})$ is not possible in our setting, since our query class **SPJUD*** explicitly forbids queries with different projections over identical subqueries, see Definition (19.6) on Page (219).

Case (2): $q = \pi_{\mathcal{A}}(q_1)$ and $\hat{q} = \hat{q}_1 \Theta \hat{q}_2$ with $\Theta \in \{\bowtie, \cup, \setminus\}$

In this subcase, we prove that

$$(\pi_{\mathcal{A}}(q_1) \neq (\hat{q}_1 \Theta \hat{q}_2)) \Rightarrow \left(\bigvee_{(t_1 \in q_1, t_1, \mathcal{A} = t)} \varphi_{q_1}^{t_1} \neq (\varphi_{\hat{q}_1}^{\hat{t}_1} \theta \varphi_{\hat{q}_2}^{\hat{t}_2}) \right)$$

is valid for an algebra operator $\Theta \in \{\bowtie, \cup, \setminus\}$ with its corresponding logical operator $\theta \in \{\wedge, \vee, \wedge \neg\}$.

Again, we make use of a contradiction to verify our claimed proposition. So, let us fix an algebra operator Θ with its associated logical operator θ and assume that

$$(\pi_{\mathcal{A}}(q_1) \neq (\hat{q}_1 \Theta \hat{q}_2)) \wedge \left(\bigvee_{(t_1 \in q_1, t_1, \mathcal{A} = t)} \varphi_{q_1}^{t_1} = (\varphi_{\hat{q}_1}^{\hat{t}_1} \theta \varphi_{\hat{q}_2}^{\hat{t}_2}) \right)$$

is true.

Since our n-ary disjunction operator \bigvee is not mapped from an algebra operator $\Theta \in \{\bowtie, \cup, \setminus\}$, there can be only one disjunctive operand in φ_q^t , if $\left(\bigvee_{(t_1 \in q_1, t_1, \mathcal{A} = t)} \varphi_{q_1}^{t_1} \right)$ should be syntactically equal to $(\varphi_{\hat{q}_1}^{\hat{t}_1} \theta \varphi_{\hat{q}_2}^{\hat{t}_2})$. Similarly to the last case, we employ our refined induction assumption and conclude that

$$(\varphi_q^t = \varphi_{q_1}^{t_1}) \Rightarrow (q \equiv q_1) \Rightarrow (\pi_{\mathcal{A}}(q_1) \equiv q_1) \Rightarrow (\mathcal{A} = \text{head}(q_1)) \not\vdash.$$

This contradicts our general assumption $\mathcal{A} \subset \text{head}(q_1)$ as formulated in Case (1).

Case (3): $q \equiv q_1 \Theta q_2$ and $\hat{q} \equiv \hat{q}_1 \hat{\Theta} \hat{q}_2$ with $\Theta, \hat{\Theta} \in \{\bowtie, \cup, \setminus\}$

According to our MAC rules of Lemma (10.1) on Page (84), we need to show

$$((q_1 \Theta q_2) \neq (\hat{q}_1 \hat{\Theta} \hat{q}_2)) \Rightarrow ((\varphi_{q_1}^{t_1} \theta \varphi_{q_2}^{t_2}) \neq (\varphi_{\hat{q}_1}^{\hat{t}_1} \hat{\theta} \varphi_{\hat{q}_2}^{\hat{t}_2}))$$

with $\Theta, \hat{\Theta} \in \{\bowtie, \cup, \setminus\}$, and $\theta, \hat{\theta} \in \{\wedge, \vee, \wedge \neg\}$. Analogously to Case (2), we contradict the negation of our claimed proposition, i.e.,

$$((q_1 \Theta q_2) \neq (\hat{q}_1 \hat{\Theta} \hat{q}_2)) \wedge ((\varphi_{q_1}^{t_1} \theta \varphi_{q_2}^{t_2}) = (\varphi_{\hat{q}_1}^{\hat{t}_1} \hat{\theta} \varphi_{\hat{q}_2}^{\hat{t}_2})).$$

We clearly see from $(\varphi_{q_1}^{t_1} \theta \varphi_{q_2}^{t_2}) = (\varphi_{\hat{q}_1}^{\hat{t}_1} \hat{\theta} \varphi_{\hat{q}_2}^{\hat{t}_2})$ that

$$(\varphi_{q_1}^{t_1} = \varphi_{\hat{q}_1}^{\hat{t}_1}) \wedge (\varphi_{q_2}^{t_2} = \varphi_{\hat{q}_2}^{\hat{t}_2}) \quad \text{as well as} \quad \Theta = \hat{\Theta}$$

have to hold. By exploiting again our reformulated induction assumption

$$(\varphi_{q_i}^t = \varphi_{\hat{q}_i}^{\hat{t}}) \Rightarrow (q_i = \hat{q}_i),$$

we infer that $q_1 = \hat{q}_1$ and $q_2 = \hat{q}_2$. This leads to

$$((\varphi_{q_1}^{t_1} \theta \varphi_{q_2}^{t_2}) = (\varphi_{\hat{q}_1}^{\hat{t}_1} \hat{\theta} \varphi_{\hat{q}_2}^{\hat{t}_2})) \Rightarrow ((q_1 = \hat{q}_1) \wedge (q_2 = \hat{q}_2) \wedge (\Theta = \hat{\Theta})) \Rightarrow ((q_1 \Theta q_2) = (\hat{q}_1 \hat{\Theta} \hat{q}_2)) \not\vdash$$

violating the assumption of our contradiction:

$$q = q_1 \Theta q_2 \quad \neq \quad \hat{q}_1 \hat{\Theta} \hat{q}_2 = \hat{q}.$$

□

Part VI

Conclusions

Chapter 21

Research questions

In this chapter, we assess our contributions in relation to our research question initially posed in Chapter (2). The following points are summarized in more detail:

- Section (21.1): an efficient lineage construction,
- Section (21.2): an orthogonal combination of optimization techniques, which are performed within the relational database layer and the probabilistic query engine, and
- Section (21.3): effective and compact data structures to represent lineage formulas within a probabilistic query engine.

21.1 Lineage construction

The first of our initial research questions asked for an alternative lineage construction approach.

Question (1): *How can we construct lineage formulas efficiently for all relational algebra queries by means of a 2-tier probabilistic database system?*

Our input queries are formulated in relational algebra. Accordingly, we can conveniently use an RDBMS for generating the set of all possible answers $Q_{\text{poss}(\mathcal{W})}$. In contrast, the corresponding answer probabilities $\mathbf{P}(t \in Q)$ with $t \in Q_{\text{poss}(\mathcal{W})}$ cannot be computed within our relational database layer, if the relational query processing part is supposed to be still polynomial in terms of database size. Instead, we put the $\#\mathcal{P}$ -hard problem of calculating $\mathbf{P}(t \in Q)$ into an additional probabilistic query engine. We exploit the well-known concept of lineage formulas in order to capture and transfer the underlying events, which are necessary to determine $\mathbf{P}(t \in Q)$.

Lineage formulas usually have very irregular forms. Such forms cannot be efficiently built within an RDBMS, because they are not supported natively. Prior to this work, systems had to apply one of following three approaches in order to build lineage formulas:

- create of nested lineage formulas within an RBDMS by restricting their lengths in advance¹ [37],
- generate a nested lineage formula within an RBDMS by setting up a network of referencing lineage subformulas [114, 101], or
- build a lineage formula in DNF within an RBDMS by forbidding relational algebra queries with difference operations [7].

In Figure (21.1) on Page (242), we list the advantages and disadvantages of these techniques.

¹Here, we assume that the maximal number of tuples of a specific relational table is conceptionally not bounded. In contrast, the length of a given tuple is considered to be limited by the cumulative size of its attribute type sizes.

Fulfillment of design goals for lineage construction						
	design goal	safe plans (MystiQ)	nested (SPROUT2)	DNF (MayBMS)	networks (PrDB/Trio)	vertical (Prophecy)
1	full relational algebra support	no	yes	no	yes	yes
2	unlimited lineage formula lengths	-	no	yes	yes	yes
3	native relational data types	yes	no	yes	yes	yes
4	result set sizes as in the deterministic case	yes	yes	yes	no	yes

Figure 21.1: Fulfillment of design goals for lineage construction

Remarkably, our vertical lineage construction approach overcomes all identified disadvantages. To do so, it exploits concepts of the relational domain calculus. Thus, the generation of the underlying domains is first performed within our relational database layer. It is strictly separated from the final construction of all lineage formulas within our probabilistic query engine. This gives us the freedom to encode and optimize domains in composed and decomposed forms as well as to seamlessly integrate lineage optimization and compression methods, see below.

Throughout this thesis, we experimentally verified that vertical lineage construction outperforms state-of-the-art approaches by one order of magnitude, if we evaluate complex tractable queries.

21.2 Orthogonal combination of lineage optimizations

In addition to lineage construction, we also studied the combination of several optimization methods.

Question (2): *How can we orthogonally combine optimization strategies performed within the relational database layer and the probabilistic query engine?*

In the past, a probabilistic database had to choose one of the following:

- a fast relational processing with an additional lineage optimization step in the probabilistic query engine, e.g., lazy plans of [82],
- a direct probability computation within the relational database layer without having an additional optimization step, e.g., safe plans of [24], or
- a compromise between the relational processing and probability computation part, e.g., hybrid plans of [82].

The characteristics of those methods are summarized in Figure (21.2) on Page (243).

Our decoupled construction of domains and lineage formulas allows us to optimize two types of query plans. They are particularly exploited within the relational database layer and the probabilistic query engine.

Most importantly, all lineage formulas are *directly* built in an optimized form independently from the query plan performed within the relational database layer. An additional lineage optimization step within our probabilistic query engine is not necessary any more.

The experiments of this work showed that an orthogonal combination of lineage optimizations outperforms existing approaches by one order of magnitude during the evaluation of fairly complex queries.

Fulfillment of design goals for lineage optimization				
	design goal	safe plans (MystiQ)	lineage transformation (MayBMS/SPROUT)	decoupled optimization (Prophecy)
1	full relational algebra support	no	no	yes
2	unconstrained relational optimizers	no	yes	yes
3	direct probability computation for tractable queries	yes	no	yes

Figure 21.2: Fulfillment of design goals for lineage optimization

Fulfillment of design goals for lineage compression				
	design goal	share plans (SPROUT2)	bisimulation (PrDB)	λ -labeling (Prophecy)
1	full relational algebra support	no	yes	yes
2	finding all shared factors (query class)	tractable SPJ	SPJUD	SPJUD* (see Def. (19.6))
3	efficient label computation	yes	no	yes
4	on-the-fly factorization	no	no	yes

Figure 21.3: Fulfillment of design goals for lineage factorization and compression

21.3 Compact data structures for lineage formulas

Last but not least, we investigated techniques for lineage factorization and compression.

Question (3): *For which query class can we identify all shared factors efficiently?*

Specifically, we were interested in identifying lineage *subformulas* that can be found multiple times within a set of lineage formulas. Shared lineage subformulas, also known as shared factors, could be easily exploited to compress our data structures used for representing lineage formulas.

Prior to this thesis, two other studies addressed lineage factorization and compression, namely factor networks [101] and shared plans [84]. The pros and cons of both frameworks are listed in Figure (21.3) on Page (243).

The first technique developed for the PrDB system is capable of identifying *all* shared factors with a quadratic complexity.

As an alternative, shared plans of SPROUT2 only works on query level. But in contrast to [101], it can only process a subset of all relational algebra queries.

We devised in this work our λ -labeling factorization and compression mechanism, which can be applied to arbitrary algebra queries. We proved for the query class **SPJUD*** with

$$\mathbf{nrSPJ} \subset \mathbf{nrSPJUD} \subset \mathbf{SPJUD}^* \subset \mathbf{SPJUD}$$

that our approach is capable of finding all existing shared factors in linear time, if we incorporate it into our vertical construction algorithm. This combination facilitates a very efficient on-the-fly construction of already compressed data structures.

Appendix A

Experimental databases

In the following, we provide a detailed description of our experimental databases and all investigated queries. The results of our experiments are discussed and presented in conjunction with the investigated concepts of Part (V). By doing so, we explain our contributions in a more self-contained manner.

This Appendix consists of the following sections:

- Section (A.1): the creation and structure of the tested probabilistic databases and
- Section (A.2): the two sets of applied queries.

To be more concrete, all our experiments are based on two probabilistic databases, which are derived from two popular data sets. We analyzed several queries on a probabilistic version of the popular *IMDB movie database*¹. We also investigated a probabilistic variant of the well-known *TPC-H benchmark database*².

In order to make our experiments as comparable as possible, we implemented and integrated all tested algorithms in our probabilistic query engine *Prophecy*. *Prophecy* is a probabilistic database framework that works in combination with a standard RDBMS. Its first prototype was implemented by Christian Winkel as part of his master thesis project [115].

To be more concrete, we utilized a local PostGres 9.4.3 database server as relational database layer in our experiments. *Prophecy* and the PostGres DBS were installed on a MacBook Pro (Retina, 15-inch, Early 2013) machine with 2.7 GHz Intel Core i7 processor and 16 GB of 1600MHz DDR3L onboard memory.

A.1 Data sets

Next, we present the creation process of the investigated IMDB and TPC-H databases in more depth.

Probabilistic database derived from IMDB movie database

Unlike the TPC-H benchmark, the IMDB movie data set neither involves a strict relational schema nor a standard set of queries. The available data files rather provide several collections of movie-related information that can be used for all kinds of applications.

For our IMDB-based experiments, we created a probabilistic database extracted from the IMDB data files available on June 2014. More specifically, we transformed the given deterministic data files into a *tuple independent database* (TID database).

¹ <http://www.imdb.com/>

² <http://www.tpc.org/tpch/>

Event relation derived from IMDB table <i>movies</i>							
$x_{\text{movie_id}}$...	x_{title}	x_{year}	x_{type}	...	$R_{\text{movies}}(\dots)$	$\mathbf{P}(e_i)$
...							
351	...	Bride of the Century	2014	tv series	...	e_{351}	0.99
352	...	Brighton Marathon	2013	tv series	...	e_{352}	0.70
353	...	Brilliant Northern Ireland	2014	tv series	...	e_{353}	0.14
...							

Implementation form of event relation derived from IMDB table <i>movies</i>							
movie_id	...	title	year	type	...	TID $R_{\text{movies}}(\dots)$	PROB $R_{\text{movies}}(\dots)$
...							
351	...	Bride of the Century	2014	tv series	...	351	0.99
352	...	Brighton Marathon	2013	tv series	...	352	0.70
353	...	Brilliant Northern Ireland	2014	tv series	...	353	0.14
...							

Figure A.1: Simplified event relations based on an extract of IMDB table *movies*

Tuple numbers of IMDB tables	
relation	#tuples
acted_in	1,704,181
actor_aka_names	113,970
actors	642,416
characters	301,607
episodes	69,434
movie_aka_titles	11,651
movie_genres	221,559
movie_keywords	144,159
movie_locations	53,625
movies	102,881

Figure A.2: Tuple counts of all tables of IMDB scenario

In a TID database (Section (4.2)), all involved atomic tuple events are assumed to be independent from each other. Therefore, the existing of a tuple does not influence the presence or the absence of any other tuple in a specific world.

The creation of such an example database is fairly simple, when we neglect the actual meaning of the introduced atomic tuple events for our test purposes. In this case, we just need to assign a unique binary random variable with a randomly chosen probability to each tuple.

Figure (A.1) on Page (246) shows two forms of a simplified extract of our probabilistic IMDB table *movies*. First, it is given as an event relation of Chapter (13). Figure (A.1) on Page (246) also depicts a more pragmatic version of our IMDB table. Particularly, we store an atomic tuple event e_{351} through its identifier 351 (column TID $R_{\text{movies}}(\dots)$) in conjunction with its probability 0.99 (column PROB $R_{\text{movies}}(\dots)$).

To sum up, the total number of all tuples generated for the IMDB test tables are shown in Figure (A.2) on Page (246).

Data set based on TPC-H benchmark database

The TPC-H benchmark is a popular test framework consisting of a business-related database schema and a suite of business oriented ad-hoc SQL queries.

In order to verify that our techniques are not restricted to a certain type of probabilistic databases, for our second test case we created a *block independent-disjoint database* (BID database). By employing the TPC-H data generator 2.14.3, we initially generated a deterministic

Event relation of TPC-H table <i>customer</i>								
x_{custkey}	...	x_{name}	$x_{\text{nationkey}}$	x_{acctbal}	...	$R_{\text{cust}}(\dots)$	$\mathbf{P}(e_{i,j})$	
⋮								
100	...	Maier	US	999	...	$e_{100,1}$	0.12	
100	...	Maier	DE	999	...	$e_{100,2}$	0.18	
100	...	Mayer	US	999	...	$e_{100,3}$	0.28	
100	...	Mayer	DE	999	...	$e_{100,4}$	0.42	
101	...	Lehman	UK	10	...	$e_{101,1}$	0.02	
101	...	Lehman	GE	10	...	$e_{101,2}$	0.08	
101	...	Lehmann	UK	10	...	$e_{101,3}$	0.18	
101	...	Lehmann	GE	10	...	$e_{101,4}$	0.72	
⋮								

Implementation form of event relation of TPC-H table <i>customer</i>								
custkey	...	name	nationkey	acctbal	...	$\text{BID}_{R_{\text{cust}}(\dots)}$	$\text{TID}_{R_{\text{cust}}(\dots)}$	$\text{PROB}_{R_{\text{cust}}(\dots)}$
⋮								
100	...	Maier	US	999	...	100	1	0.12
100	...	Maier	DE	999	...	100	2	0.18
100	...	Mayer	US	999	...	100	3	0.28
100	...	Mayer	DE	999	...	100	4	0.42
101	...	Lehman	UK	10	...	101	1	0.02
101	...	Lehman	GE	10	...	101	2	0.08
101	...	Lehmann	UK	10	...	101	3	0.18
101	...	Lehmann	GE	10	...	101	4	0.72
⋮								

Figure A.3: Simplified event relation based on an extract of TPC-H table *customer*

TPC-H database of scale factor 0.032. We used this traditional relational database to build our tested BID database by exploiting two additional concepts called *event keys* and *tuple blocks* [111].

In short, a specific block contains all tuples that share *same values* for a certain set of attributes. Those attributes are given by the event key of a relation. In our experiments, we set all event keys to the classical relational keys provided by the original TPC-H schema.

We know from Section (4.2) that all tuples *within* a block are associated with a set of *disjoint* atomic tuple events. Accordingly, only one tuple of each block can exist in a given world, since they mutually exclude themselves. On the other hand, tuple events from different blocks are still independent from each other.

The importance of BID databases arises from their capability of expressing attribute uncertainty. In that context, the attributes of the event key are considered as *certain*. In contrast, the remaining attributes are classified as *uncertain*. Then, the values for all certain (event key) attributes in a specific block are always identical and the uncertain (non-event-key) attributes have different values. Accordingly, a block can also be considered as a single meta tuple with several possible values for its uncertain attributes.

Example A.1 (Attribute uncertainty expressed by a BID table). *To give an example for a BID table, let us consider the TPC-H table customer (Figure (A.3) on Page (247)) having the event key custkey and the uncertain attributes name, nationkey, acctbal, Concretely, we are interested in a single customer described in form of a meta tuple t with the event key value custkey = 100. For this meta tuple, we presume that a data integration tool provides two possible values for the attributes name and nationkey, and one unique value for the attribute acctbal:*

$$t_{100} = (\underbrace{100}_{\text{custkey}}, \underbrace{\{\text{Maier}, \text{Mayer}\}}_{\text{name}}, \underbrace{\{\text{US}, \text{DE}\}}_{\text{nationkey}}, \underbrace{\{999\}}_{\text{acctbal}}, \dots).$$

Obviously, our given meta tuple t_{100} violates the first normal form of the relational data model [74]. It cannot be directly stored in a classical RDBMS. Hence, we flatten t_{100} and capture the inherent attribute uncertainty by defining a block of four tuples given in the first normal form. They represent all possible attribute value combinations of name and nationkey:

$$\begin{aligned} t_{100,1} &= (100, \text{Maier}, \text{US}, 999, \dots)_{(\text{custkey}, \text{name}, \text{nationkey}, \text{acctbal}, \dots)} \\ t_{100,2} &= (100, \text{Maier}, \text{DE}, 999, \dots)_{(\text{custkey}, \text{name}, \text{nationkey}, \text{acctbal}, \dots)} \\ t_{100,3} &= (100, \text{Mayer}, \text{US}, 999, \dots)_{(\text{custkey}, \text{name}, \text{nationkey}, \text{acctbal}, \dots)} \\ t_{100,4} &= (100, \text{Mayer}, \text{DE}, 999, \dots)_{(\text{custkey}, \text{name}, \text{nationkey}, \text{acctbal}, \dots)}. \end{aligned}$$

Each tuple is annotated with its own atomic tuple event $e_{100,1}, \dots, e_{100,4}$, see Figure (A.3) on Page (247). In contrast to a TID database, tuple events are not always independent. All tuple events within a given block are defined as disjoint. Accordingly, only one of the given four value combinations $t_{100,1}, \dots, t_{100,4}$ can exist in reality.

To determine the initial probability of a specific atomic tuple event $\mathbf{P}(e_{i,j})$, we combine the probabilities given for its uncertain attribute values. If we assume that our data integration tool annotates all possible attribute values of t_{100} with

$$\begin{aligned} \mathbf{P}(\text{custkey} = 100) &= 1.0 \\ \mathbf{P}(\text{name} = \text{Maier}) &= 0.3 \\ \mathbf{P}(\text{name} = \text{Mayer}) &= 0.7 \\ \mathbf{P}(\text{nationkey} = \text{US}) &= 0.4 \\ \mathbf{P}(\text{nationkey} = \text{DE}) &= 0.6 \\ \mathbf{P}(\text{acctbal} = 999) &= 1.0, \end{aligned}$$

then we calculate the probabilities of our introduced block tuples to

$$\begin{aligned} t_{100,1} &= (100, \text{Maier}, \text{US}, 999) \quad \text{with} \quad \mathbf{P}(e_{100,1}) = 1.0 * 0.3 * 0.4 * 1.0 = 0.12 \\ t_{100,2} &= (100, \text{Maier}, \text{DE}, 999) \quad \text{with} \quad \mathbf{P}(e_{100,2}) = 1.0 * 0.3 * 0.6 * 1.0 = 0.18 \\ t_{100,3} &= (100, \text{Mayer}, \text{US}, 999) \quad \text{with} \quad \mathbf{P}(e_{100,3}) = 1.0 * 0.7 * 0.4 * 1.0 = 0.28 \\ t_{100,4} &= (100, \text{Mayer}, \text{DE}, 999) \quad \text{with} \quad \mathbf{P}(e_{100,4}) = 1.0 * 0.7 * 0.6 * 1.0 = 0.42. \end{aligned}$$

In our case, the values of different attributes are assumed to be independent.

Figure (A.3) on Page (247) shows a simplified extract of the table customer discussed earlier as event table and a more practical variant. For instance, it encodes the atomic tuple event $e_{100,1}$ by two identifiers 100 and 1 standing for its block number, see column $BID_{R_{\text{cust}}}(\dots)$, and its tuple identifier within that block (column $TID_{R_{\text{cust}}}(\dots)$). Moreover, its probability $\mathbf{P}(e_{100,1}) = 0.12$ is stored (column $PROB_{R_{\text{cust}}}(\dots)$).

In contrast to Example (A.1) on Page (247), we picked only one uncertain attribute per original table. For this attribute, we introduced a block expressing three possible attribute values for each original TPC-H tuple. This construction procedure yielded to final table sizes shown in Figure (A.4) on Page (249). They are approximately three times larger than the original sizes.

A.2 Tested Queries

Subsequently, we present the queries which have been examined in our four experiment series of Part (V):

- Section (13.5): relational processing part,
- Section (15.2): lineage construction within probabilistic query engine,

Sizes of generated TPC-H tables	
relation	#tuples
customer	14,400
lineitem	577,395
nation	75
orders	144,000
part	19,200
partsupp	76,200
supplier	15
region	960

Figure A.4: Tuple counts of all relations for TPC-H test database

- Section (17.3): lineage optimization and probability computation and
- Section (19.5): lineage factorization and compression.

The main focus of work lies on the construction of lineage formulas. Therefore, our experiments concentrated on *tractable* test queries, see Definition (5.1) on Page (29). By that, we could efficiently evaluate the corresponding lineage formulas by Lemma (6.3) on Page (40). The effort of creating lineage formulas was then dominating the costs of computing the respective probabilities and the benefits of our methods are directly visible on the overall query processing times.

Nevertheless, we emphasize that *hard* queries, see Definition (5.1) on Page (29), can also benefit from our ideas and techniques, e.g., hard queries with an approximative probability computation, when their evaluation is based on lineage formulas.

IMDB queries

The official IMDB movie data set does not include any pre-defined queries. So, we formulated six non-trivial queries for our tests. The short forms of our six IMDB queries IMDB-Q1, ..., IMDB-Q6 are listed in Figure (A.5) on Page (250). In order to obtain concise notations, we made use of a mapping between our IMDB tables and a set of abbreviated relation identifiers and attribute labels.

Example A.2 (Short form of example query). *Our first query IMDB-Q1 has the abbreviated form*

$$\pi_{B,C}((R_1(A,B) \bowtie R_2(A)) \bowtie R_3(A,C)).$$

It stands for the query

$$\begin{aligned} & \pi_{title_1, title_2}((\rho_{title_1 \leftarrow title}(\pi_{movie_id, title}(\sigma_{(gender=female)}(movies))) \bowtie \\ & \pi_{movie_id}(\sigma_{(location <> 'Los Angeles, California, USA'}(movie_location))) \bowtie \\ & \rho_{title_2 \leftarrow title}(\pi_{movie_id, title}(movie_aka_titles))) \end{aligned}$$

Most significantly, our query selection covered a broad range of relational algebra queries from **SPJUD**. As shown in Figure (A.5) on Page (250), we took special care of including all types of algebra operators and join types.

TPC-H queries

For our TPC-H scenario, we chose seven queries denoted as TPCH-Q1, ..., TPCH-Q7. Analogously to our IMDB queries, we took a combination of all operators and join types into account. The core structure of our TPC-H queries is shown in Figure (A.6) on Page (251).

Our first two queries TPCH-Q1 and TPCH-Q2 are directly inferred from the conjunctive core of the first and seventh original TPC-H query. In [82], Olteanu et al. tested their SPROUT

Tested IMDB queries		
query	short form	
IMDB-Q1	$\pi_{B,C}((R_1(A,B) \bowtie R_2(A)) \bowtie R_3(A,C))$	
IMDB-Q2	$\pi_{A,B}(((R_1(A,B) \bowtie R'_1(B)) \bowtie R_3(A)) \bowtie R_4(A))$	
IMDB-Q3	$\pi_A(((R_1(A) \bowtie R_2(A)) \bowtie R_3(A)) \bowtie R_4(A))$	
IMDB-Q4	$(R_1(A) \bowtie R_2(A)) \cup \pi_A((R'_1(A) \bowtie R_3(A,B)) \bowtie R_4(B))$	
IMDB-Q5	$((R_1(A) \bowtie R_2(A)) \bowtie R_3(A,B)) \setminus \pi_{A,B}((R_4(A,B,C) \bowtie R_5(C)) \bowtie R_6(A))$	
IMDB-Q6	$((R_1(A) \cup R_2(A)) \setminus R_3(A)) \bowtie \pi_A((R_4(A,B) \bowtie R_5(B)) \bowtie R_6(A))$	

Relations based on IMDB tables		
query	IMDB table	basic query
IMDB-Q1	$R_1(A,B)$ $R_2(A)$ $R_3(A,C)$	$\rho_{title_1} \leftarrow title(\pi_{movie_id, title}(\sigma_{(gender=female)}(movies)))$ $\pi_{movie_id}(\sigma_{(location <> 'Los Angeles, California, USA')}(movie_location))$ $\rho_{title_2} \leftarrow title(\pi_{movie_id, title}(movie_aka_titles))$
IMDB-Q2	$R_1(A,B)$ $R'_1(B)$ $R_3(A)$ $R_4(A)$	$\pi_{movie_id, prod_year}(\sigma_{(type='movie' \wedge title \text{ like 'A\%'})}(movies))$ $\pi_{prod_year}(\sigma_{(type='tv series' \wedge prod_year \geq 2013)}(movies))$ $\pi_{movie_id}(\sigma_{(genre='Comedy')}(movie_genres))$ $\pi_{movie_id}(\sigma_{(location='Los Angeles, California, USA')}(movie_locations))$
IMDB-Q3	$R_1(A)$ $R_2(A)$ $R_3(A)$ $R_4(A)$	$\pi_{movie_id}(acted_in)$ $\pi_{movie_id}(\sigma_{(keyword \in \{'murder','death','blood'\})}(movie_keywords))$ $\pi_{movie_id}(\sigma_{(prod_year \geq 2013)}(movies))$ $\pi_{movie_id}(\sigma_{(genre='Drama')}(movie_genres))$
IMDB-Q4	$R_1(A)$ $R_2(A)$ $R'_1(A)$ $R_3(A,B)$ $R_4(B)$	$\pi_{actor_id}(\sigma_{(gender='male')}(actors))$ $\pi_{actor_id}(\sigma_{(name \text{ like 'A\%'})}(actor_aka_names))$ $\pi_{actor_id}(\sigma_{((gender='female') \wedge (actor_id < 100))}(actors))$ $\pi_{actor_id, character_id}(\sigma_{(billing_pos < > 2)}(acted_in))$ $\pi_{character_id}(\sigma_{(name <> 'Himself')}(characters))$
IMDB-Q5	$R_1(A)$ $R_2(A)$ $R_3(A,B)$ $R_4(A,B,C)$ $R_5(C)$ $R_6(A)$	$\pi_{movie_id}(\sigma_{(prod_year=2014)}(movies))$ $\pi_{movie_id, (}(\sigma_{(genre='Comedy')}(movie_genre))$ $\pi_{movie_id, episode_id}(\sigma_{(season_nr=1)}(episodes))$ $\pi_{movie_id, episode_id, character_id}(acted_in)$ $\pi_{character_id}(\sigma_{(name <> 'Himself')}(characters))$ $\pi_{movie_id}(\sigma_{(title \text{ like 'Z\%'})}(movie_aka_titles))$
IMDB-Q6	$R_1(A)$ $R_2(A)$ $R_3(A)$ $R_4(A,B)$ $R_5(B)$ $R_6(A)$	$\pi_{movie_id}(\sigma_{(type='movie')}(movies))$ $\pi_{movie_id}(\sigma_{(keyword='independent-film')}(movie_keywords))$ $\pi_{movie_id, episode_id}(\sigma_{(location='New York City, New York, USA')}(movie_locations))$ $\pi_{movie_id, character_id}(\sigma_{(billing_pos=1)}(acted_in))$ $\pi_{character_id}(characters)$ $\pi_{movie_id}(\sigma_{(genre='Western')}(movie_genres))$

Properties of IMDB queries			
query	class	(1:m)-joins	(m:n)-joins
IMDB-Q1	nrSPJ	yes	no
IMDB-Q2	SPJ	yes	no
IMDB-Q3	nrSPJ	yes	yes
IMDB-Q4	SPJU	yes	yes
IMDB-Q5	nrSPJD	yes	yes
IMDB-Q6	nrSPJUD	yes	yes

Figure A.5: Tested IMDB queries with short forms and properties

algorithm by only using the conjunctive cores of all original TPC-H queries. Unfortunately, those queries only involve join operations of type (1:m). Accordingly, the computed result sizes are always bounded by their largest input relations. In fact, all queries tested in [82] only select and project tuples from their largest input relations.

More challenging queries also include (m:n)-joins, which multiply the sizes of the given input tables. To study this class of queries as well, we added the queries TPC-H-Q3 and TPC-H-Q4 to our selection of queries. Moreover, we also investigated the queries TPC-H-Q5, TPC-H-Q6, and TPC-H-Q7 as representatives for the queries with union and difference operations. The properties

Tested TPC-H queries	
query	short form
TPCH-Q1	$\pi_{B,C,D}(R_1(A) \bowtie R_2(A,B,C,D)) \bowtie R_3(B)$
TPCH-Q2	$\pi_{A,C,\dots,H}(R_4(A,B) \bowtie R_1(B,C,\dots,H)) \bowtie \pi_C(R_2(C,I) \bowtie R_3(I))$
TPCH-Q3	$(\pi_B(R_1(A) \bowtie R_2(A,B)) \bowtie R_3(B,C)) \bowtie \pi_C(R_4(D) \bowtie R_5(C,D))$
TPCH-Q4	$\pi_B((R_1(A) \bowtie R_2(A,B)) \bowtie R_3(A)) \bowtie \pi_B(R_4(C) \bowtie R_5(B,C))$
TPCH-Q5	$\pi_B(R_1(A) \bowtie R_3(A,B) \bowtie R_2(B)) \cup \pi_B(R_4(B,C) \bowtie R_5(C))$
TPCH-Q6	$R_1(A) \bowtie (\pi_B(R_2(A,B) \bowtie R_3(A)) \setminus \pi_B(R_4(C,B) \bowtie R_5(C)))$
TPCH-Q7	$\pi_A(R_1(B) \bowtie R_2(A,B)) \setminus (\pi_A(R_3(C) \bowtie R_4(A,C)) \cup \pi_A(R_5(A,D) \bowtie R_6(D)))$

Relations based on TPC-H tables		
query id	short form	basic query
TPCH-Q1	$R_1(A)$ $R_2(A,B,C,D)$ $R_3(B)$	$\pi_{\text{custkey}}(\sigma(\text{mktsegment}='BUILDING')(\text{customer}))$ $\pi_{\text{custkey}, \text{orderkey}, \text{orderdate}, \text{shippriority}}(\sigma(\text{orderdate}<1992-01-10)(\text{orders}))$ $\pi_{\text{orderkey}}(\sigma(\text{shipdate}>1992-01-10)(\text{lineitem}))$
TPCH-Q2	$R_1(B,C,\dots,H)$ $R_2(C,I)$ $R_3(I)$ $R_4(A,B)$	$\pi_{\text{nationkey}, \text{custkey}, \dots, \text{comment}}(\text{customer})$ $\pi_{\text{textcustkey}, \text{orderkey}}(\sigma(\text{orderdate} \in \{1993-01-01, \dots, 1995-3-09\})(\text{orders}))$ $\pi_{\text{orderkey}}(\sigma(\text{returnflag}='N')(\text{lineitem}))$ $\pi_{\text{name}, \text{nationkey}}(\text{nation})$
TPCH-Q3	$R_1(A)$ $R_2(A,B)$ $R_3(B,C)$ $R_4(D)$ $R_5(C,D)$	$\pi_{\text{partkey}}(\text{partsupp})$ $\pi_{(\text{partkey}, \text{suppkey})}(\sigma(\text{commitdate}<1996-01-01)(\text{lineitem}))$ $\pi_{\text{suppkey}, \text{nationkey}}(\sigma((\text{mktsegment}='BUILDING') \wedge (\text{nationkey}=1))(\text{supplier}))$ $\pi_{\text{custkey}, \text{nationkey}}(\sigma(\text{custkey}<750)(\text{customer}))$ $\pi_{\text{custkey}}(\text{orders})$
TPCH-Q4	$R_1(A)$ $R_2(A,B)$ $R_3(A)$ $R_4(C)$ $R_5(B,C)$	$\pi_{\text{partkey}}(\text{part})$ $\pi_{\text{partkey}, \text{suppkey}}(\text{partsupp})$ $\pi_{\text{partkey}}(\text{lineitem})$ $\pi_{\text{nationkey}}(\sigma((\text{custkey}<1600) \wedge (\text{nationkey}=1))(\text{customer}))$ $\pi_{\text{suppkey}, \text{nationkey}}(\sigma(\text{suppkey}<800)(\text{supplier}))$
TPCH-Q5	$R_1(A)$ $R_2(B)$ $R_3(A,B)$ $R_4(B,C)$ $R_5(C)$	$\pi_{\text{nationkey}}(\text{customer})$ $\pi_{\text{suppkey}}(\text{partsupp})$ $\pi_{\text{nationkey}, \text{suppkey}}(\text{supplier})$ $\pi_{\text{suppkey}, \text{orderkey}}(\text{lineitem})$ $\pi_{\text{orderkey}}(\text{orders})$
TPCH-Q6	$R_1(A)$ $R_2(A,B)$ $R_3(A)$ $R_4(C,B)$ $R_5(C)$	$\pi_{\text{suppkey}}(\sigma((\text{shipmode}='MAIL') \wedge (\text{receiptdate}<1992-03-01) \wedge (\text{returnflag}='R'))(\text{lineitem}))$ $\pi_{\text{partkey}}(\text{part})$ $\pi_{\text{partkey}, \text{suppkey}}(\text{partsupp})$ $\pi_{\text{nationkey}, \text{suppkey}}(\text{supplier})$ $\pi_{\text{nationkey}}(\text{customer})$
TPCH-Q7	$R_1(B)$ $R_2(A,B)$ $R_3(C)$ $R_4(A,C)$ $R_5(A,D)$ $R_6(D)$	$\pi_{\text{nationkey}}(\text{customer})$ $\pi_{\text{suppkey}, \text{nationkey}}(\sigma(\text{acctbal}<-700)(\text{supplier}))$ $\pi_{\text{partkey}}(\text{part})$ $\pi_{\text{suppkey}, \text{partkey}}(\text{partsupp})$ $\pi_{\text{suppkey}, \text{orderkey}}(\text{lineitem})$ $\pi_{\text{orderkey}}(\text{orders})$

Properties of TPC-H queries			
query	class	(1:m)-joins	(m:n)-joins
TPCH-Q1	nrSPJ	yes	no
TPCH-Q2	nrSPJ	yes	no
TPCH-Q3	nrSPJ	yes	yes
TPCH-Q4	nrSPJ	yes	yes
TPCH-Q5	nrSPJU	yes	yes
TPCH-Q6	nrSPJD	yes	yes
TPCH-Q7	nrSPJUD	yes	yes

Figure A.6: Tested TPC-H queries with short forms and properties

of our seven test queries are listed in Figure (A.6) on Page (251).

Appendix B

Basic notations

In this chapter, we clarify some basic notations used in this work.

B.1 Restriction of tuples

In principle, we define a tuple t over a domain \mathbf{D} , i.e., $t \in \mathbf{D}$. This domain is built as a cartesian product of attribute or variable domains $\text{dom}(A_i)$ and $\text{dom}(x_i)$:

$$\mathbf{D}_{\mathcal{A}} := \text{dom}(A_1) \times \dots \times \text{dom}(A_n) \quad \text{or} \quad \mathbf{D}_{\bar{x}} := \text{dom}(x_1) \times \dots \times \text{dom}(x_n),$$

where $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\bar{x} = \{x_1, \dots, x_n\}$ are our given sets of attributes and variables, see also Definition (10.1) on Page (80).

We often consider a *restricted* tuple \hat{t} , which is derived from a tuple $t \in \mathbf{D}$. Such a tuple only contains values for a certain attribute or variable subset with $\mathcal{B} \subseteq \mathcal{A}$ or $\bar{y} \subseteq \bar{x}$. In those cases, we also write $\hat{t} = t_{\mathcal{B}}$ or $\hat{t} = t_{\bar{y}}$ indicating the restriction/projection of t to the values of \mathcal{B} or \bar{y} :

$$((\hat{t} = t_{\mathcal{B}}) :\Leftrightarrow (\{\hat{t}\} = \pi_{\mathcal{B}}(\{t\}))) \quad \text{or} \quad ((\hat{t} = t_{\bar{y}}) :\Leftrightarrow (\{\hat{t}\} = \pi_{\bar{y}}(\{t\}))).$$

For a better reading, we often add the attributes or variables of the underlying domain to a restricted tuple. In the literature, the notation $t[\bar{y}]$ also describes a restriction of t . We instead spare the usage of squared brackets for describing the classes of a partition induced by an equivalence relation.

Example B.1 (Restricted tuples). *Let $\bar{x} = \{x_1, x_2, x_3\}$ be a set of variables with the variable domains*

$$\text{dom}(x_1) = \text{dom}(x_2) = \text{dom}(x_3) = \mathbb{N}$$

and the overall domain

$$\mathbf{D}_{\bar{x}} = \text{dom}(x_1) \times \text{dom}(x_2) \times \text{dom}(x_3).$$

Using the variable subsets

$$\bar{y}_1 := \{x_1\}, \quad \bar{y}_2 := \{x_2, x_3\}, \quad \text{and} \quad \bar{y}_3 := \emptyset,$$

we can restrict the tuple $t = (1, 2, 3)$ as follows:

$$t_{\bar{y}_1} = (1), \quad t_{\bar{y}_2} = (2, 3), \quad \text{and} \quad t_{\bar{y}_3} = ().$$

In the case of writing only tuple values, we often add the variables they are defined over:

$$(1)_{\bar{y}_1}, \quad (2, 3)_{\bar{y}_2}, \quad \text{and} \quad ().$$

B.2 Overlapping concatenation of tuples

Our work takes advantage of a special operation for concatenating two tuples. This operation requires that the values from both operands are identical, when they are defined over the same attributes/variables. Otherwise, we cannot apply our overlapping concatenation.

Definition B.1 (Overlapping concatenation of tuples). *Let*

$$t = (c_1, \dots, c_m) \in \mathbf{D}_{\bar{x}} \quad \text{and} \quad \hat{t} = (\hat{c}_1, \dots, \hat{c}_n) \in \mathbf{D}_{\bar{y}}$$

be two tuples defined over the two overlapping sets of variables \bar{x} and \bar{y} . If we split the overlapping variable set $(\bar{x} \cup \bar{y})$ into three partitions

$$\bar{z}_1 := \bar{x} \setminus \bar{y}, \quad \bar{z}_2 := \bar{x} \cap \bar{y}, \quad \text{and} \quad \bar{z}_3 := \bar{y} \setminus \bar{x},$$

we define the overlapping concatenation operation $t \bullet \hat{t}$ as a binary function

$$\bullet : \{(t, \hat{t}) \in (\mathbf{D}_{\bar{x}} \times \mathbf{D}_{\bar{y}}) \mid t_{\bar{z}_2} = \hat{t}_{\bar{z}_2}\} \rightarrow \mathbf{D}_{(\bar{x} \cup \bar{y})},$$

where

$$\forall z \in (\bar{x} \cup \bar{y}) : (t \bullet \hat{t})_z := \begin{cases} t_z & \text{if } (z \in (\bar{z}_1 \cup \bar{z}_2)) \\ \hat{t}_z & \text{else.} \end{cases}$$

Example B.2 (Overlapping concatenation of tuples). *The overlapping concatenation of the two tuples*

$$t_1 = (1, 2)_{(x_A, x_B)} \quad \text{and} \quad t_2 = (2, 3, 4)_{(x_B, x_C, x_D)}$$

is determined to

$$t = (t_1 \bullet t_2) := ((1, 2)_{(x_A, x_B)} \bullet (2, 3, 4)_{(x_B, x_C, x_D)}) = (1, 2, 3, 4)_{(x_A, x_B, x_C, x_D)}.$$

On the contrary, an overlapping concatenation is not defined between the tuples

$$t_1 = (1, 2)_{(x_A, x_B)} \quad \text{and} \quad t_3 = (3, 4, 5)_{(x_B, x_C, x_D)},$$

since $(2)_{x_B} \neq (3)_{x_B}$.

B.3 Disjoint union of sets

A *disjoint union* operation involving the sets \mathcal{M}_1 and \mathcal{M}_2 that have no elements in common is denoted as:

$$(\mathcal{M}_3 = \mathcal{M}_1 \dot{\cup} \mathcal{M}_2) :\Leftrightarrow ((\mathcal{M}_3 = \mathcal{M}_1 \cup \mathcal{M}_2) \wedge (\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset)).$$

Example B.3 (Disjoint union of sets). *The sets $\mathcal{M}_1 = \{1, 2\}$ and $\mathcal{M}_2 = \{3, 4\}$ can be combined by a disjoint union operation: $\mathcal{M}_1 \dot{\cup} \mathcal{M}_2 = \{1, 2, 3, 4\}$. On the contrary, the sets $\mathcal{M}_3 = \{1, 2\}$ and $\mathcal{M}_4 = \{2, 3\}$ are obviously not disjoint, because $\mathcal{M}_3 \cap \mathcal{M}_4 = \{2\} \neq \emptyset$.*

Bibliography

- [1] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. *SIGMOD Rec.*, 16(3):34–48, December 1987.
- [2] Charu C. Aggarwal. Probabilistic querying and mining of biological images.
- [3] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB Conf.*, pages 1151–1154, 2006.
- [4] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 496–505, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [5] Periklis Andritsos, Ariel Fuxman, and Rene J. Miller. Clean answers over dirty databases: A probabilistic approach. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 30. IEEE Computer Society, 2006.
- [6] L. Antova, C. Koch, and D. Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1479–1480, April 2007.
- [7] Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *ICDE Conf.*, pages 983–992, 2008.
- [8] Subi Arumugam, Ravi Jampani, Luis Leopoldo Perez, Fei Xu, Christopher M. Jermaine, and Peter J. Haas. Mcdbr: Risk analysis in the database. *PVLDB*, 3(1):782–793, 2010.
- [9] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowl. and Data Eng.*, 4:487–502, October 1992.
- [10] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.
- [11] Omar Benjelloun, Anish Das Sarma, Alon Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, March 2008.
- [12] George Beskales, Mohamed A. Soliman, Ihab F. Ilyas, and Shai Ben-David. Modeling and querying possible repairs in duplicate detection. *PVLDB*, 2(1):598–609, 2009.
- [13] Garrett Birkhoff. Lattice theory. In *Colloquium Publications*, volume 25. Amer. Math. Soc., 3. edition, 1967.
- [14] Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Automatically building probabilistic databases from the web. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 185–188, New York, NY, USA, 2011. ACM.

- [15] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Probabilistic models to reconcile complex data from inaccurate data sources. In Barbara Pernici, editor, *CAiSE*, volume 6051 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2010.
- [16] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *VLDB*, pages 71–81. Morgan Kaufmann, 1987.
- [17] Mou-Yen Chen, Amlan Kundu, and Jian Zhou. Off-line handwritten word recognition using a hidden markov model type stochastic network. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):481–496, 1994.
- [18] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1112–1127, September 2004.
- [19] Reynold Cheng and Sunil Prabhakar. Managing uncertainty in sensor database. *SIGMOD Rec.*, 32(4):41–46, December 2003.
- [20] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
- [21] E. F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD Rec.*, 15(4):53–53, December 1986.
- [22] E F Codd. More commentary on missing information in relational databases (applicable and inapplicable information). *SIGMOD Rec.*, 16(1):42–50, March 1987.
- [23] Nilesh Dalvi and Philip Bohannon. Robust web extraction: an approach based on a probabilistic tree-edit model. *ACM SIGMOD international conference on Management of data*, 2009.
- [24] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, October 2007.
- [25] Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.
- [26] Nilesh N. Dalvi, Karl Schnaitter, and Dan Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214, 2010.
- [27] Professor Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
- [28] A. Das Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE Conf.*, pages 1023–1032, April 2008.
- [29] Landon Detwiler, Wolfgang Gatterbauer, Brenton Louie, Dan Suciu, and Peter Tarczy-Hornoch. Integrating and ranking uncertain scientific data. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 1235–1238. IEEE, 2009.
- [30] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, September 1996.
- [31] Xin Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 687–698. VLDB Endowment, 2007.

- [32] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *IEEE 29th International Conference on Data Engineering (ICDE 2013), Brisbane, Australia*, Brisbane, Australia, 2013. IEEE Computer Society.
- [33] Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz, and Andreas Zoefle. Querying uncertain spatio-temporal data. In Anastasios Kementsietsidis and Marcos Antonio Vaz Salles, editors, *ICDE*, pages 354–365. IEEE Computer Society, 2012.
- [34] Anderson A. Ferreira, Marcos Andre Gonzalves, and Alberto H. F. Laender. A brief survey of automatic methods for author name disambiguation. *SIGMOD Record*, 41(2):15–26, 2012.
- [35] Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *Proc. VLDB Endow.*, 5(5):490–501, January 2012.
- [36] Robert Fink and Dan Olteanu. A dichotomy for non-repeating queries with negation in probabilistic databases. In *PODS*, pages 144–155, 2014.
- [37] Robert Fink, Dan Olteanu, and Swaroop Rath. Providing support for full relational algebra in probabilistic databases. In *ICDE*, pages 315–326, 2011.
- [38] Norbert Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of the 16th International Conference on Very Large Data Bases, VLDB '90*, pages 696–707, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [39] Norbert Fuhr and Thomas Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. IS*, 15(1):32–66, 1997.
- [40] Avigdor Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2006.
- [41] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.
- [42] Minos N. Garofalakis, Kurt P. Brown, Michael J. Franklin, Joseph M. Hellerstein, Daisy Zhe Wang, Eirinaios Michelakis, Liviu Tancau, Eugene Wu 0002, Shawn R. Jeffery, and Ryan Aipperspach. Probabilistic data management for pervasive computing: The data furnace project. *IEEE Data Eng. Bull.*, 29(1):57–63, 2006.
- [43] Erol Gelenbe and Georges Hebrail. A probability model of uncertainty in data bases. In *Proceedings of the Second International Conference on Data Engineering*, pages 328–333, Washington, DC, USA, 1986. IEEE Computer Society.
- [44] S P Ghosh. Statistical relational tables for statistical database management. *IEEE Trans. Softw. Eng.*, 12(12):1106–1116, December 1986.
- [45] Gista Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.
- [46] Gösta Grahne. Dependency satisfaction in databases with incomplete information. In Umeshwar Dayal, Gunter Schlageter, and Lim Huat Seng, editors, *Tenth International Conference on Very Large Data Bases, August 27-31, 1984, Singapore, Proceedings*, pages 37–45. Morgan Kaufmann, 1984.
- [47] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In Umeshwar Dayal, Kyu Y. Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang K. Cha, Young K. Kim, Umeshwar Dayal, Kyu Y. Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang K. Cha, and Young K. Kim, editors, *VLDB*, pages 965–976. ACM, 2006.

- [48] Mena B. Habib and Maurice van Keulen. Improving toponym disambiguation by iteratively enhancing certainty of extraction. In Ana L. N. Fred, Joaquim Filipe, Ana L. N. Fred, and Joaquim Filipe, editors, *KDIR*, pages 399–410. SciTePress, 2012.
- [49] Oktie Hassanzadeh and Renée J. Miller. Creating probabilistic databases from duplicated data. *The VLDB Journal*, 18(5):1141–1166, October 2009.
- [50] Ihab F. Ilyas and Mohamed A. Soliman. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on DM. Morgan & Claypool, 2011.
- [51] Tomasz Imieliński and Witold Lipski. Incomplete Information in Relational Databases. *J. ACM*, 31(4):761–791, September 1984.
- [52] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *Proc. VLDB Endow.*, 3(1-2):429–438, September 2010.
- [53] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter J. Haas. The monte carlo database system: Stochastic analysis close to the data. *ACM Trans. Database Syst.*, 36(3):18:1–18:41, August 2011.
- [54] Ravi Jampani, Fei Xu, Mingxi Wu, Luis L. Perez, Christopher Jermaine, and Peter J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 687–700, New York, NY, USA, 2008. ACM.
- [55] Abhay Kumar Jha and Dan Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, pages 249–261, 2012.
- [56] Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3):403–440, 2013.
- [57] Sanjeev Khanna, Sudeepa Roy, and Val Tannen. Queries with difference on probabilistic databases. *PVLDB*, 4(11):1051–1062, 2011.
- [58] Nodira Khousainova, Magdalena Balazinska, and Dan Suciu. Probabilistic event extraction from rfid data. In Gustavo Alonso, Jose A. Blakeley, and Arbee L. P. Chen, editors, *ICDE*, pages 1480–1482. IEEE, 2008.
- [59] Christoph Koch. Approximating predicates and expressive queries on probabilistic databases. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, pages 99–108, New York, NY, USA, 2008. ACM.
- [60] Christoph Koch. MayBMS: A System for Managing Large Uncertain and Probabilistic Databases. In *Managing and Mining Uncertain Data*, ch. 6. Springer-Verlag, 2008.
- [61] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.
- [62] Arun Kumar and Christopher Ré. Probabilistic management of ocr data using an rdbms. *Proc. VLDB Endow.*, 5(4):322–333, December 2011.
- [63] Michel Lacroix and Alain Pirotte. Domain-oriented relational languages. In *VLDB*, pages 370–378. IEEE Computer Society, 1977.
- [64] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

- [65] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Proview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, September 1997.
- [66] S. L. Lauritzen and D. J. Spiegelhalter. Readings in uncertain reasoning. chapter Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems, pages 415–448. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [67] Sebastian Lehrack. Applying weighted queries on probabilistic databases. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM '12*, pages 2209–2213, New York, NY, USA, 2012. ACM.
- [68] Sebastian Lehrack, Sascha Saretz, and Ingo Schmitt. QSQLp: Eine Erweiterung der probabilistischen Many-World-Semantik um Relevanzwahrscheinlichkeiten. In *BTW*, pages 494–513, 2011.
- [69] Sebastian Lehrack, Sascha Saretz, and Ingo Schmitt. QSQL2: Query Language Support for Logic-Based Similarity Conditions on Probabilistic Databases. In *RCIS*, pages 1–12, 2012.
- [70] Sebastian Lehrack, Sascha Saretz, and Christian Winkel. Proqua: a system for evaluating logic-based scoring functions on uncertain relational data. In *EDBT Conf.*, pages 761–764, 2013.
- [71] Sebastian Lehrack and Ingo Schmitt. QSQL: Incorporating Logic-Based Retrieval Conditions into SQL. In Hiroyuki Kitagawa, Yoshiharu Ishikawa, Qing Li, and Chiemi Watanabe, editors, *DASFAA (1)*, volume 5981 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2010.
- [72] Sebastian Lehrack and Ingo Schmitt. A probabilistic interpretation for a geometric similarity measure. In *Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'11*, pages 749–760, Berlin, Heidelberg, 2011. Springer-Verlag.
- [73] Leonid Libkin and Limsoon Wong. Semantic representations and query languages for or-sets. In Catriel Beeri, editor, *PODS*, pages 37–48. ACM Press, 1993.
- [74] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.
- [75] Robert Mandelbaum, G. Kamberova, and Max Mintz. Stereo depth estimation: A confidence interval approach. In *ICCV*, pages 503–509, 1998.
- [76] Anan Marie and Avigdor Gal. Managing uncertainty in schema matcher ensembles. In Henri Prade and V. S. Subrahmanian, editors, *SUM*, volume 4772 of *Lecture Notes in Computer Science*, pages 60–73. Springer, 2007.
- [77] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.
- [78] Tadeusz Morzy, Theo Härder, and Robert Wrembel, editors. *Advances in Databases and Information Systems - 16th East European Conference, ADBIS 2012, Poznań, Poland, September 18-21, 2012. Proceedings*, volume 7503 of *Lecture Notes in Computer Science*. Springer, 2012.
- [79] Andrew Nierman and H. V. Jagadish. Protodb: Probabilistic data in xml. In *VLDB*, pages 646–657. Morgan Kaufmann, 2002.

- [80] Nuria Oliver, Barbara Rosario, and Alex Pentland. Statistical modeling of human interactions. In *In CVPR Workshop on Interpretation of Visual Motion*, pages 39–46. IEEE, 1998.
- [81] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 389–402, New York, NY, USA, 2009. ACM.
- [82] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, Washington, DC, USA, 2009. IEEE Computer Society.
- [83] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.
- [84] Dan Olteanu and Hongkai Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE Conf.*, ICDE '12, pages 282–293, Washington, DC, USA, 2012. IEEE Computer Society.
- [85] Fabian Panse. *Duplicate Detection in Probabilistic Relational Databases*. dissertation, University of Hamburg, 2014.
- [86] Fabian Panse, Maurice van Keulen, and Norbert Ritter. Indeterministic handling of uncertain decisions in deduplication. *J. Data and Information Quality*, 4(2):9, 2013.
- [87] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
- [88] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. K-nearest neighbors in uncertain graphs. *Proc. VLDB Endow.*, 3(1-2):997–1008, September 2010.
- [89] Disheng Qiu, Paolo Papotti, and Lorenzo Blanco. Future locations prediction with uncertain data. In Hendrik Blockeel, Kristian Kersting, and Siegfried Nijssen, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8188 of *Lecture Notes in Computer Science*, pages 417–432. Springer Berlin Heidelberg, 2013.
- [90] Erhard Rahm, Philip A. Bernstein, and Jayant Madhavan. A Survey of Approaches to Automatic Schema Matching. *VLDB*, 10(4):334–350, September 2001.
- [91] Christopher Re, Nilesch N. Dalvi, and Dan Suciu. Query evaluation on probabilistic databases. *IEEE Data Eng. Bull.*, 29(1):25–31, 2006.
- [92] Christopher Re, Nilesch N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.
- [93] Christopher Ré and Dan Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.
- [94] Christopher Re and Dan Suciu. Managing Probabilistic Data with MystiQ: The Can-Do, the Could-Do, and the Can't-Do. In *SUM*, pages 5–18, 2008.
- [95] Robert Ross, V. S. Subrahmanian, and John Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, January 2005.
- [96] Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.
- [97] Ingo Schmitt. Weighting in CQQL. BTU Cottbus, Computer Science Reports 04/07, 2007.

- [98] Ingo Schmitt. QQL: A DB&IR Query Language. *The VLDB Journal*, 17(1):39–56, 2008.
- [99] Ingo Schmitt and Gunter Saake. A comprehensive database schema integration method based on the theory of formal concepts. *Acta Inf.*, 41(7-8):475–524, 2005.
- [100] Prithviraj Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 596–605, April 2007.
- [101] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18:1065–1090, October 2009.
- [102] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.
- [103] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.
- [104] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne E. Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *SIGMOD Conference*, pages 1239–1242, 2008.
- [105] Sarvjeet Singh, Chris Mayfield, Rahul Shah, Sunil Prabhakar, Susanne Hambrusch, Jennifer Neville, and Reynold Cheng. Database support for probabilistic attributes and tuples. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE '08*, pages 1053–1061, Washington, DC, USA, 2008. IEEE Computer Society.
- [106] Yannis Sismanis, Ling Wang, Ariel Fuxman, Peter J. Haas, and Berthold Reinwald. Resolution-aware query answering for business intelligence. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 976–987. IEEE, 2009.
- [107] Mohamed A. Soliman, Ihab F. Ilyas, and Mina Saleeb. Building ranked mashups of unstructured sources with uncertain information. *Proc. VLDB Endow.*, 3:826–837, 2010.
- [108] Abhinav Srivastava, Amlan Kundu, Shamik Sural, and Arun Majumdar. Credit card fraud detection using hidden markov model. *IEEE Transactions on Dependable and Secure Computing*, 5(1):37–48, 2008.
- [109] Michael Stonebraker. The postgres dbms. In Hector Garcia-Molina and H. V. Jagadish, editors, *SIGMOD Conference*, page 394. ACM Press, 1990. SIGMOD Record 19(2), June 1990.
- [110] Dan Suci, Andrew J. Connolly, and Bill Howe. Embracing uncertainty in large-scale computational astrophysics. In *Proceedings of the Third VLDB workshop on Management of Uncertain Data (MUD2009) in conjunction with VLDB 2009, Lyon, France, August 28th, 2009.*, pages 63–77, 2009.
- [111] Dan Suci, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases. Synthesis Lectures on Data Management*. Morgan & Claypool Publishers, 2011.
- [112] Thanh T. L. Tran, Charles A. Sutton, Richard Cocci, Yanming Nie, Yanlei Diao, and Prashant J. Shenoy. Probabilistic inference over rfid streams in mobile environments. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 1096–1107. IEEE, 2009.
- [113] Daisy Zhe Wang, Eirinaios Michelakis, Michael J. Franklin, Minos N. Garofalakis, and Joseph M. Hellerstein. Probabilistic declarative information extraction. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 173–176. IEEE, 2010.

- [114] Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *Managing and Mining Uncertain Data*, pages 113–148. Springer, 2008.
- [115] Christian Winkel. Assessment of Efficient Query Processing Algorithms on Probabilistic Databases. Master’s thesis, BTU Cottbus, 2014.
- [116] Catherine Wolfram. Strategic bidding in a multi-unit auction: An empirical analysis of bids to supply electricity in England and Wales. *RAND Journal of Economics*, 29:703–725, 1998.
- [117] Fei Xu, Kevin S. Beyer, Vuk Ercegovic, Peter J. Haas, and Eugene J. Shekita. E = mc3: managing uncertain enterprise data in a cluster-computing environment. In *SIGMOD Conference*, pages 441–454. ACM, 2009.
- [118] Sze Man Yuen, Yufei Tao, Xiaokui Xiao, Jian Pei, and Donghui Zhang. Superseding nearest neighbor search on uncertain spatial databases. *IEEE Trans. Knowl. Data Eng.*, 22(7):1041–1055, 2010.
- [119] Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proc. VLDB Endow.*, 6(9):757–768, July 2013.
- [120] Esteban Zimányi. Query evaluation in probabilistic relational databases. *Theor. Comput. Sci.*, 171(1-2):179–219, January 1997.
- [121] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Finding top-k maximal cliques in an uncertain graph. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 649–652. IEEE, 2010.